МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп’ютерних інформаційних технологій
Кафедра кібербезпеки

ОСІДАК Владеслав Ігорович

Алгоритми динамічного аналізу поведінки шкідливого програмного забезпечення / Algorithms for Dynamic Analysis of Malware Behavior

спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
В. І. Осідак

____

Науковий керівник
к.т.н., доцент С.В.Івасьєв

____

Кваліфікаційну роботу допущено
до захисту:

«____» __________ ___2025 р.

Завідувач кафедри
________________ В.В.Яцків

ТЕРНОПІЛЬ - 2025

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 - Кібербезпека та захист інформації
освітньо-професійна програма –Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри
________________ В.В.Яцків
«____» ____________ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ОСІДАКУ ВЛАДЕСЛАВУ ІГОРОВИЧУ
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми динамічного аналізу поведінки шкідливого програмного забезпечення / Algorithms for Dynamic Analysis of Malware Behavior
керівник роботи д.т.н., доцент С.В. Івасьєв
затверджені наказом по університету від 20 грудня 2024 року № 938
2. Строк подання студентом закінченої випускної кваліфікаційної роботи 5 грудня 2025 року.
3. Вихідні дані до кваліфікаційної роботи: завдання на випускну кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
– провести аналіз сучасних підходів до класифікації та виявлення шкідливого ПЗ, визначивши основні напрями розвитку методів аналізу загроз;
– дослідити характеристики різних типів шкідливого ПЗ, узагальнити способи їх дії та принципи обходу систем захисту;
– розробити методичні засади побудови моделі виявлення шкідливого ПЗ на основі аналізу його поведінкових і структурних ознак;
– виконати експериментальну перевірку запропонованої моделі, оцінити її точність і сформулювати рекомендації щодо підвищення ефективності аналізу шкідливих програм.
5. Перелік графічного матеріалу у роботі:
– Схема розповсюдження троянів-завантажувачів .
– Схема атаки із використанням кейлогера.
– Схема роботи методу градієнтного бустингу .
– Схема роботи методу Бройдена-Флетчера-Голдфарба-Шанно .
– Схема роботи методу логістичної регресії .
– 3D-розподіл атрибутів навчального набору даних.
– Мережева візуалізація сильних кореляцій між атрибутами навчальної вибірки .

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Теоретичні засади аналізу шкідливого програмного забезпечення	12.2024 р. – 03.2025 р.	
2	Методологічні основи побудови моделі виявлення шкідливого програмного забезпечення	03.2025 р. – 06.2025 р.	
3	Програмна реалізація та експериментальна перевірка результатів	06.2025 р. – 11.2025 р.	

Студент ______________ Владеслав ОСІДАК
(підпис)

Керівник роботи _____________ к.т.н., доцент Степан ІВАСЬЄВ
(підпис)

АНОТАЦІЯ

Осідак В.І. Алгоритми динамічного аналізу поведінки шкідливого програмного забезпечення - Рукопис.

Дослідження на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека та захист інформації», освітньо-професійна програма «Кібербезпека». – Західноукраїнський національний університет, Тернопіль, 2025.

Проаналізовано сучасні підходи до виявлення та класифікації шкідливого ПЗ, проведено огляд типів загроз і принципів їх дії. Розроблено алгоритмічну модель на основі методу Бройдена–Флетчера–Голдфарба–Шанно та реалізовано програмний модуль у середовищі Visual Studio 2022 мовою C#. Проведено експериментальні дослідження з використанням реального набору даних із ресурсу Kaggle, що підтвердили точність моделі на рівні 87,69 % та високу ефективність її застосування у реальному середовищі.

Розроблена система дозволяє автоматизувати процеси виявлення шкідливого ПЗ, забезпечуючи швидкий аналіз поведінкових характеристик програм і зниження ризиків кібератак. Результати роботи можуть бути використані у системах моніторингу безпеки, антивірусних рішеннях і навчальних комплексах з кіберзахисту.

КЛЮЧОВІ СЛОВА: ДИНАМІЧНИЙ АНАЛІЗ, ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МАШИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ, ML.NET, КІБЕРБЕЗПЕКА.

ANNOTATION

Osidak V.I. Algorithms for Dynamic Analysis of Malware Behavior - Manuscript.

Research for the degree of Master in specialty 125 "Cybersecurity and information protection", educational and professional program "Cybersecurity". - Western Ukrainian National University, Ternopil, 2025.

Modern approaches to detecting and classifying malicious software were analyzed, a review of threat types and principles of their action was conducted. An algorithmic model was developed based on the Broyden–Fletcher–Goldfarb–Shannot method and a software module was implemented in the Visual Studio 2022 environment in C#. Experimental studies were conducted using a real dataset from the Kaggle resource, which confirmed the accuracy of the model at 87.69% and the high efficiency of its application in a real environment.

The developed system allows to automate the processes of detecting malicious software, providing quick analysis of behavioral characteristics of programs and reducing the risks of cyberattacks. The results of the work can be used in security monitoring systems, antivirus solutions and training complexes on cyber defense.

KEYWORDS: DYNAMIC ANALYSIS, MALWARE, MACHINE LEARNING, CLASSIFICATION, ML.NET, CYBERSECURITY.

ЗМІСТ

ВСТУП.. 6
1 ТЕОРЕТИЧНІ ЗАСАДИ АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.. 8
1.1 Огляд сучасних підходів до класифікації та аналізу шкідливого ПЗ.......... 8
1.2 Характеристика основних типів шкідливого програмного забезпечення та способів його дії 11
1.3 Аналіз існуючих систем автоматичного виявлення шкідливого ПЗ 16
1.4 Постановка задачі.. 24
2 МЕТОДОЛОГІЧНІ ОСНОВИ ПОБУДОВИ МОДЕЛІ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 26
2.1 Вибір методу машинного навчання для виявлення ПЗ.............................. 26
2.2 Вибір та опис набору даних для навчання і тестування моделі................. 35
2.3 Розроблення алгоритму машинного навчання для класифікації ПЗ 46
2.4 Визначення показників оцінки якості моделі 52
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА РЕЗУЛЬТАТІВ.. 55
3.1 Вибір мови програмування, середовища розробки та бібліотеки.............. 55
3.2 Побудова основних алгоритмів системи 62
3.3 Реалізація ПЗ для навчання, збереження та використання моделі 65
3.4 Проведення експериментів на основі реальних зразків............................. 74
3.5 Аналіз отриманих результатів, оцінка ефективності моделі та формування практичних рекомендацій 79
ВИСНОВКИ.. 83
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 86
ДОДАТОК А. Додаткові діаграми аналізу тренувального набору даних.......... 92
ДОДАТОК Б. Лістинги програмного коду 99
ДОДАТОК В. Кпії публікацій……………………………………………115

ВСТУП

Актуальність роботи. Зі зростанням масштабів цифровізації суспільства та інтеграції інформаційних технологій у всі сфери діяльності людства різко зросла кількість шкідливого програмного забезпечення (ПЗ), здатного порушувати роботу комп'ютерних систем і завдавати значних збитків. Традиційні антивірусні засоби, засновані на сигнатурному виявленні, дедалі частіше виявляються неефективними проти нових типів загроз, які постійно змінюють власну структуру, використовують методи обфускації та шифрування, а також адаптуються до особливостей операційного середовища. Цей виклик потребує переходу від реактивних до проактивних підходів, заснованих на інтелектуальному аналізі поведінки програм.

Методи машинного навчання створюють передумови для розроблення систем, здатних виявляти невідомі раніше зразки шкідливого коду шляхом аналізу їхніх поведінкових характеристик та системних параметрів. У порівнянні з класичними технологіями, такі рішення забезпечують гнучкість і здатність до самонавчання, що особливо важливо в умовах динамічного кіберпростору. Проте залишаються відкритими проблеми вибору інформативних ознак, підвищення точності класифікації та адаптації моделей до реальних умов експлуатації.

Мета роботи полягає у розробленні інтелектуальної системи виявлення шкідливого програмного забезпечення на основі методів машинного навчання, здатної автоматично класифікувати програмні об'єкти за поведінковими характеристиками.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз сучасних підходів до класифікації та виявлення шкідливого ПЗ, визначивши основні напрями розвитку методів аналізу загроз;

– дослідити характеристики різних типів шкідливого ПЗ, узагальнити способи їх дії та принципи обходу систем захисту;

– розробити методичні засади побудови моделі виявлення шкідливого ПЗ на основі аналізу його поведінкових і структурних ознак;

– виконати експериментальну перевірку запропонованої моделі, оцінити її точність і сформулювати рекомендації щодо підвищення ефективності аналізу шкідливих програм.

Об’єкт дослідження – процес аналізу та виявлення шкідливого програмного забезпечення в інформаційних системах.

Предмет дослідження – методи та принципи класифікації програмних процесів за ознаками їх поведінки з метою розпізнавання шкідливих дій.

Наукова новизна одержаних результатів визначається наступним чином:

– запропоновано узагальнену модель аналізу шкідливого ПЗ, що базується на комплексному використанні поведінкових і статистичних характеристик програмних процесів, що дозволяє підвищити точність виявлення прихованих загроз;

– сформульовано підхід до вибору інформативних ознак для класифікації програмних об’єктів, який забезпечує скорочення обчислювальних витрат і покращення стабільності результатів під час аналізу нових зразків шкідливого коду.

Практична цінність одержаних результатів полягає в тому, що:

– розроблено узагальнений підхід до автоматизованого аналізу шкідливого програмного забезпечення, який може бути використаний для підвищення ефективності систем кіберзахисту;

– реалізовано концепцію побудови програмного модуля, здатного виконувати класифікацію програмних процесів за ознаками їх поведінки та підтримувати адаптацію до нових типів загроз.

Публікації та апробація кваліфікаційної роботи.

1. Осідак В. Івасьєв С. Онлайн засоби динамічного аналізу шкідливого програмного забезпечення / Матеріали науков-практичного симпозіуму «ЗАХИСТ ІНФОРМАЦІЇ», Тернопіль, 2025. – С. 56-61.

2. Осідак В. Поведінковий аналіз у задачі виявлення шкідливих програм / Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології»(КБКІТ-2025), Тернопіль, 2025. - С. 57-60.

1 ТЕОРЕТИЧНІ ЗАСАДИ АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд сучасних підходів до класифікації та аналізу шкідливого ПЗ

Шкідливе програмне забезпечення – це програми, які здійснюють небажані втручання у роботу комп’ютерів або пристроїв, незалежно від того, чи надано на це згоду [1]. Такі програми виконують руйнівні дії у комп’ютерних системах, завдаючи істотної шкоди зараженим машинам. Оскільки шкідливе ПЗ є ключовим компонентом у багатьох кібератаках, їхні завдання зазвичай зосереджені на порушенні будь-якого з трьох базових атрибутів інформації – конфіденційності, цілісності чи доступності даних [2]. В останні роки збитки від діяльності шкідливого ПЗ стали настільки значними, що в наукових та бізнесових виданнях повідомляють про втрати на рівні трильйонів доларів для світової економіки.

Впродовж останніх десятиліть технології зазнали стрімкого розвитку, що кардинально змінило спосіб нашого життя й підходи до цифровізації. Поширення інтернет-пристроїв, розвиток хмарних сервісів, застосування методів машинного навчання та активне зростання соціальних мереж створили складну цифрову екосистему. Хоча вона принесла багато переваг – від глобальної взаємодії до масштабних бізнес-операцій – вона водночас породила нові вектори загроз і вразливостей, що атакують конфіденційність, цілісність та доступність інформації.

З початком пандемії кількість нових зразків шкідливого ПЗ значно зросла – за оцінками, до 2020 року з’явилося близько 1,5 мільярда нових зразків. Цей бум змусив організації активізувати свої заходи безпеки [3]. Шкідливе ПЗ класифікують за поведінкою, функціональністю й способами поширення.

Кібератаки спрямовані не лише на окремих користувачів, а й на підприємства різного масштабу та галузей, і їхні наслідки можуть бути катастрофічними – від викрадення персональних і фінансових даних до значних збоїв у роботі ІТ-інфраструктури та тяжких репутаційних втрат. Компанії

постійно протистоять еволюції загроз, зокрема програмам-вимагачам, фішинговим схемам та вразливостям у ланцюгах постачання ПЗ, що вимагає безперервного вдосконалення заходів захисту та оперативного реагування [4].

Завдання захисту нових технологій відноситься до сфери кібербезпеки – міждисциплінарного напряму, що охоплює методи, інструменти й процеси захисту інформаційних систем і даних. Протягом останніх років цей напрямок істотно розвинувся: поруч з класичними засобами введено аналітичні та поведінкові підходи, а також методи машинного навчання, які показують обіцяну здатність виявляти складні та нові варіанти атак [5].

Як ілюстрація наукового інтересу та практичної потреби, кількість публікацій, присвячених шкідливому ПЗ у контексті кібербезпеки, зросла експоненційно в період 2010–2024 років (рис. 1.1). Це вказує на необхідність розробки нових методів і методологій для подолання сучасних викликів; систематичні огляди підтверджують, що шкідливе програмне забезпечення залишається однією з головних загроз для сучасних технологій [6]. Важливим чинником успішності атак є винахідливість зловмисників у комбінуванні тактик обходу захисту, а також людський фактор, який часто визначає шлях поширення загроз [7].

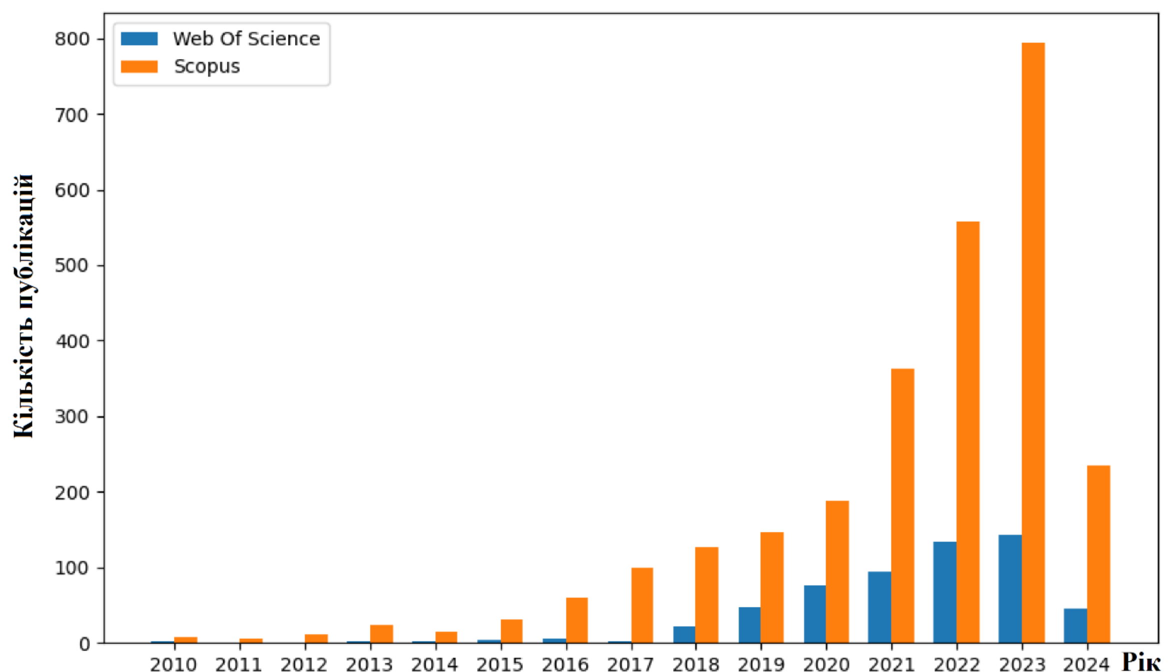

Рисунок 1.1 –Кількість публікацій про шкідливе ПЗ в галузі кібербезпеки

Методи машинного навчання відіграють фундаментальну роль у виявленні шкідливого ПЗ та розвитку відповідних технологій захисту. У міру зростання складності кібератак дедалі важче ефективно протидіяти загрозам, покладаючись виключно на традиційні правила та сигнатури; саме тут машинне навчання набуває ключового значення, оскільки дозволяє створювати алгоритми й моделі, здатні самостійно виявляти закономірності й адаптуватися до нових видів атак [8].

Системи виявлення на основі машинного навчання можуть опрацьовувати великі обсяги даних і виявляти складні патерни та аномалії, які залишаються непоміченими при класичному статичному аналізі. Завдяки механізмам навчання на прикладах такі моделі послідовно підвищують свою здатність розпізнавати шкідливі зразки в міру надходження нових даних, що покращує їхню точність і адаптивність у практичних умовах експлуатації [9].

У методологічному вимірі підходи на базі машинного навчання поділяються на кілька основних категорій: методи з навчанням під наглядом (supervised), без нагляду (unsupervised) та напівавтоматичні/посилені підходи (semi-/reinforcement). Для задачі класифікації шкідливого ПЗ домінують supervised-алгоритми (наприклад, ансамблеві методи, SVM, логістична регресія) і глибинні нейромережі (CNN, RNN, трансформери), тоді як unsupervised-підходи застосовуються для виявлення аномалій і нових небезпечних патернів у відсутності міток. Вибір класу алгоритмів обґрунтовується характером представлених ознак, наявністю маркованих даних та вимогами до реал-таємності й продуктивності [10].

Ключовим компонентом будь-якої системи є етап вилучення ознак. У контексті шкідливого ПЗ практикується три основні типи представлень: статичні (аналітика бінарних парсерів, метадані файлу), динамічні (трасування системних викликів, мережевих з’єднань, змін у файловій системі) та гібридні комбінації обох видів. Якість та інформативність цих ознак безпосередньо зумовлюють здатність моделі узагальнювати знання на нові сімейства загроз [11].

Разом із перевагами існує низка практичних викликів. По-перше, проблема дисбалансу класів і зашумленості датасетів ускладнює навчання та оцінювання

моделей; по-друге, явище «concept drift» (зміна розподілу даних у часі) вимагає механізмів безперервного перенавчання або online-навчання; по-третє, атаки, спрямовані на моделі (adversarial attacks), ставлять питання про стійкість і надійність класифікаторів. Додатково, у виробничому середовищі критичними є такі нефункціональні показники, як час затримки рішення, частота хибних спрацьовувань (false positives) і трактованість моделей для аналітиків безпеки.

Оцінювання ефективності методів має спиратися на широкий набір метрик (Precision, Recall, F1, AUC-ROC, PR-AUC) і репрезентативні набори тестових даних, що відображають реальні умови експлуатації; при цьому бажано застосовувати узгоджені протоколи валідації та відкриті критерії відтворюваності експериментів. Невирішені експериментальні питання і обмежена порівнюваність результатів у численних публікаціях підкреслюють потребу в уніфікації методик дослідження [12].

Отже, інтеграція машинного навчання у системи виявлення шкідливого ПЗ відкриває значні перспективи для підвищення адаптивності та проактивності кіберзахисту; водночас для переходу від дослідницьких прототипів до практично надійних рішень необхідні комплексні підходи, що поєднують якісні ознаки, стійкі алгоритми, механізми постійного оновлення моделей та оцінку в реальних сценаріях.

1.2 Характеристика основних типів шкідливого програмного забезпечення та способів його дії

Шкідливе ПЗ не просто «заражає» систему – воно проходить певні етапи життєвого циклу. Спочатку відбувається первинне проникнення (інфекційний вектор), яке може бути організовано через шкідливі електронні листи, фішингові посилання, незахищені USB-носії або вразливі веб-сайти [13]. Потім програма намагається підвищити привілеї, отримати права адміністратора й закріпити свою присутність, застосовуючи техніки приховування, метаморфізму та поліфорізму. Деякі приклади включають приховування у легітимних процесах

(process hollowing), підробку системних бібліотек, шифрування або упаковку коду. Отже, шкідливе ПЗ виконує основну функцію – збирання даних, шифрування файлів, спостереження за користувачем чи участь у ботнеті – та, за необхідності, підтримує зв’язок із керівними серверами для отримання нових інструкцій.

Вірус – один з найстаріших різновидів шкідливого ПЗ. Згідно з Cisco, вірус є програмою, що самореплікується, вставляючи свій код у інші програми і активується при запуску зараженого файлу; він може спричинити від невеликих перешкод до серйозної втрати даних та виходу системи з ладу [14]. Поширення відбувається через виконувані файли, скрипти, електронні листування чи знімні носії. Захист включає регулярне оновлення систем, використання антивірусів та виявлення невідповідностей у контрольних суммах файлів.

Черв’яки – це автономні програми, здатні самостійно розповсюджуватись мережею без участі користувача, використовуючи сканування хостів, експлуатацію вразливостей мережевих сервісів або слабкі облікові дані [15]. Вони можуть містити додаткові модулі (завантажувачі, бекдори, компоненти для формування ботнету) і, крім прямого інфікування, створювати надлишкове навантаження на мережеві ресурси в результаті масового трафіку. Через ці властивості черв’яки представляють серйозну загрозу для цілісності та доступності локальних і корпоративних мереж.

На рис. 1.2 наведено спрощену схему поширення комп’ютерних черв’яків у мережі з підключенням до провайдера.

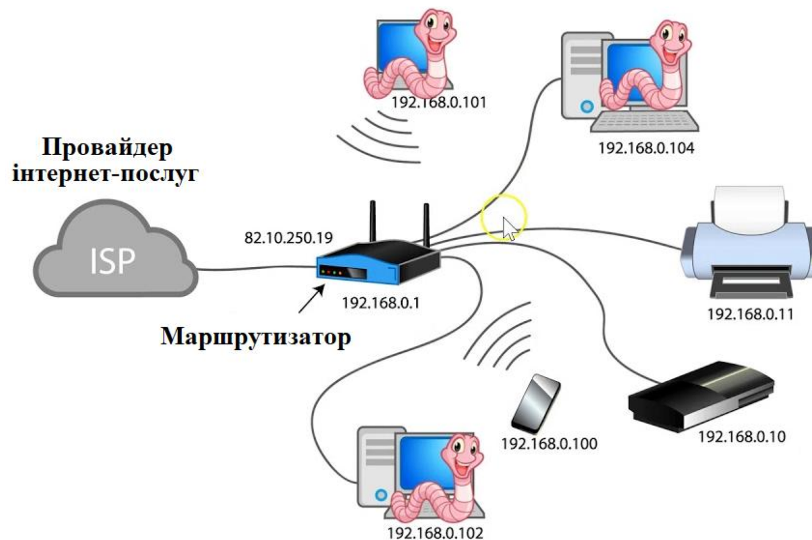

Рисунок 1.2 –Схема поширення комп'ютерних черв’яків

Навіть один вразливий вузол у мережі може стати початком ланцюгового зараження, коли шкідливий код поступово поширюється між пристроями, використовуючи спільні ресурси та відкриті з’єднання. Неконтрольоване розповсюдження здатне порушити стабільність роботи всієї системи, спричинивши збої в роботі обладнання, втрату даних і деградацію пропускної здатності мережі.

Троянські програми – це шкідливі компоненти, які маскуються під легітимні додатки або приховані у вигляді корисного вмісту, та виконують на компрометованому хості небажані дії без явної взаємодії користувача [16]. Зазвичай вони не мають здатності до саморозповсюдження як черв’яки, але забезпечують точку входу для завантаження додаткових модулів, встановлення бекдорів або організації каналів управління і передачі даних. Через багатоступеневу архітектуру та використання прихованих каналів троянці утруднюють виявлення та класифікацію на ранніх етапах проникнення.

На рис. 1.3 наведено типову багатоступеневу схему поширення та функціонування троян-завантажувачів, яка ілюструє основні етапи життєвого циклу такого загрозливого вектора.

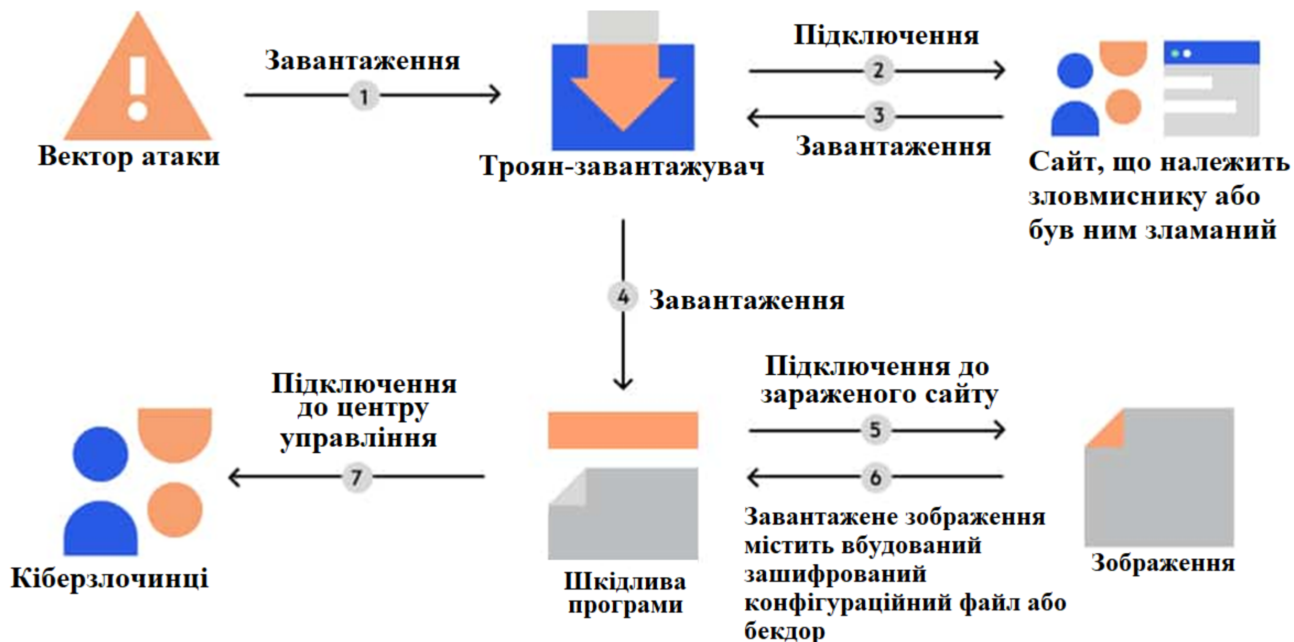

Рисунок 1.3 – Схема розповсюдження троянів-завантажувачів [17]

Із зовнішнього вектора атаки відбувається першочергове завантаження завантажувача, який встановлюється на цільовий пристрій і встановлює мережеве з’єднання для отримання подальших компонентів; далі завантажувач підключається до веб-ресурсу, що належить зловмиснику або був компрометований, і отримує основний шкідливий модуль. У деяких випадках проміжний елемент – зображення або інший носій – містить вбудовану, зашифровану конфігурацію або бекдор, що витягається та інтерпретується на цільовому вузлі; це служить способом стеганографічного приховування команд і конфігурацій. Після завантаження основного компоненту відбувається встановлення каналу управління і передачі, через який кіберзлочинці можуть віддалено керувати інфікованою машиною, передавати додаткові модулі або ексфільтрувати дані. Схема відображає також можливість багаторазових циклів, що забезпечують гнучкість та стійкість атаки.

Шпигунське ПЗ – категорія загроз, спрямована на приховане збори та передачу конфіденційної інформації (логінів, паролів, вмісту клавіатурних натискань, знімків екрана, даних буфера обміну тощо) без відома власника пристрою [18]. Воно характеризується тривалим, малошумним перебуванням у системі та орієнтацією на розвідку й довготривалу компрометацію приватних або корпоративних ресурсів. Частим варіантом шпигунського ПЗ є кейлогери,

які фіксують введення з клавіатури й супутню активність для отримання облікових даних.

На рис. 1.4 зображено типову схему атаки із застосуванням кейлогера.

Рисунок 1.4 – Схема атаки із використанням кейлогера [19]

Початковий вектор атаки може містити соціально-інженерні елементи (фішинг, шкідливі вкладення) або приховані завантажувачі, які доставляють і встановлюють кейлогер на цільовому вузлі; після інсталяції агент реєструє натискання клавіш, знімає скриншоти й опрацьовує буфер обміну, зберігаючи зібрані дані локально у тимчасових файлах; для зниження виявлення дані часто шифруються або пакуються та періодично відправляються на віддалений сервер керування через масковані канали (HTTP/S, приховані DNS-запити або інші тунелі); зловмисник отримує доступ до ексфільтрованої інформації, аналізує її і, за потреби, ініціює подальші дії (експлуатацію облікових записів, рух латеральними шляхами, ексфільтрацію додаткових даних). Схема також демонструє використання проміжних носіїв (наприклад, компрометованих веб-ресурсів або мультимедійних файлів із вбудованими конфігураціями) для прихованої доставки команд і оновлень.

Шпигунське ПЗ реалізує довгострокову розвідувальну функцію в ланцюжку атаки, створюючи умовну «тіньову» інфраструктуру доступу до цінних даних, що робить його особливо небезпечним для витончених, таргетованих атак і підкреслює необхідність дослідження методів кореляції багато джерельних телеметрій для адекватної ідентифікації таких інцидентів.

Нові покоління шкідливого ПЗ поєднують різні техніки: приховування (stealth), шифрування, мультиплатформеність та використання штучного інтелекту. Наприклад, триразова екстернація (triple extortion) у програмах-вимагачах передбачає не лише шифрування даних, а й крадіжку інформації та погрози розголошення, що змушує жертв платити викуп.

Перспективи розвитку захисту включають:

– розробку евристичних та поведінкових моделей, які досліджують потоки даних та взаємодії системних компонентів у реальному часі.

– використання квантових обчислень та гомоморфного шифрування для безпечної обробки даних у хмарних сервісах.

– формування нормативної бази для криміналізації кіберзлочинів, міжнародної співпраці та обміну інформацією.

Шкідливе програмне забезпечення стало невід’ємною складовою кіберзагроз. Класифікація за типами та поведінкою дозволяє систематизувати знання про різновиди шкідливого ПЗ і розробити ефективні стратегії захисту. Аналіз шкідливого ПЗ включає статичні, динамічні та гібридні методи, доповнені машинним і глибинним навчанням, що забезпечує детекцію нових, складних загроз. Розуміння принципів дії різноманітних класів шкідливого ПЗ допомагає будувати системи виявлення та розробляти політику реагування.

1.3 Аналіз існуючих систем автоматичного виявлення шкідливого ПЗ

Актуальність проблеми автоматичного виявлення шкідливого ПЗ зумовлена не стільки кількістю окремих інцидентів, скільки зміною природи загроз: сучасні зразки поєднують стеганографію, модульну архітектуру, мережеві засоби розповсюдження та механізми самозахисту, що істотно ускладнює їхню ідентифікацію традиційними інструментами. У таких умовах ефективність засобів виявлення визначається не лише їхньою спроможністю розпізнати відому сигнатуру, але й здатністю аналізувати поведінкові патерни,

корелювати багатопоточні телеметричні потоки і адаптуватися до еволюції шкідливих сімейств.

Парадигма сучасних систем детекції дедалі більше орієнтується на інтеграцію різнорідних джерел даних і на застосування статистичних та навчальних методів для виявлення аномалій, що не вкладаються в попередньо відомі моделі поведінки. Водночас інтеграція інтелектуальних компонентів породжує нові виклики: забезпечення репрезентативності навчальних вибірок, стійкість моделей до обфускації та adversarial-атак, а також проблема «concept drift» – непередбачуваної зміни розподілу ознак у часі.

У практичній площині ці виклики трансформуються в серйозні обмеження: високий рівень хибних тривог знижує придатність рішення в операційних умовах, надмірні обчислювальні витрати ускладнюють масштабування, а закритість промислових систем ускладнює оцінювання перенесення моделей між середовищами. Саме тому аналіз існуючих систем повинен спиратися на критичну оцінку як їхніх детекційних можливостей, так і здатності зберігати працездатність та довіру в реальних експлуатаційних сценаріях.

SentinelOne – це комплексна платформа захисту кінцевих точок і розширеного виявлення загроз, призначена для попередження, виявлення та автоматизованого реагування на сучасні кіберзагрози у реальному часі (рис. 1.5). Система орієнтована на забезпечення проактивного захисту кінцевих пристроїв, кореляцію телеметрії та скорочення часу від реагування до усунення інциденту.

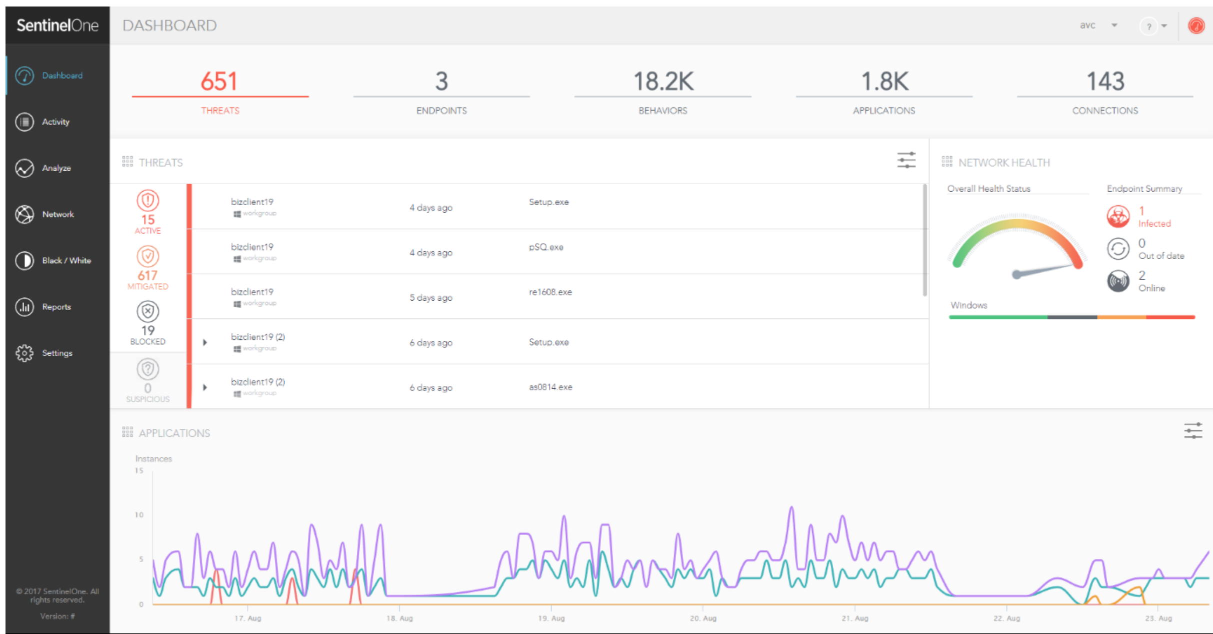

Рисунок 1.5 – Приклад інтерфейсу системи «SentinelOne» [20]

Архітектурно рішення базується на тонкому агента на кожній кінцевій точці, який збирає локальну телеметрію, виконує первинну обробку та застосовує вбудовані алгоритми машинного навчання й поведінковий аналіз для миттєвої блокування підозрілих дій; зібрані дані синхронізуються з централізованим хмарним або локально розміщеним сервером аналітики, де відбувається кореляція подій з урахуванням глобальної інформації про загрози, історії поведінки і сигнатур [21]. Керування здійснюється через єдину консолі управління, яка надає панелі моніторингу, маршрути автоматичного реагування, інструменти для розслідування інцидентів і інтеграційні API для SOC/ SIEM-систем; у складі платформи реалізовані механізми автоматичного відновлення після шкідливих дій і засоби підтримки оперативного полювання за загрозами, що дозволяє поєднувати автоматичну протидію та ручне розслідування в єдиному робочому процесі.

Перевагами даного інструменту є:

– автоматизований захист у реальному часі на основі поведінкового аналізу та вбудованих моделей машинного навчання, що дозволяє блокувати підозрілі дії без очікування сигнатур;

– можливість повної телеметрії кінцевих точок – централізована кореляція подій, інтегрований інструментарій розслідування й консолідована панель керування для SOC-операцій [22];

– механізми автономного реагування й відновлення (rollback/ remediation), які скорочують час реагування й дозволяють швидше повернутися до робочого стану після інциденту [23];

– хмарна/гібридна архітектура зі здатністю масштабувати аналітику та оперативно оновлювати репутаційні дані про загрози, що підвищує адаптивність до нових сімейств ПЗ.

До недоліків належать:

– відносно висока вартість ліцензування та експлуатації для середніх і малих організацій, що може обмежувати широке розгортання [24];

– ризик хибних спрацьовувань і додаткове навантаження на аналітичні процеси SOC при некоректно налаштованих політиках, що вимагає тонкого тюнінгу;

– часткова залежність від хмарних сервісів і зовнішньої телеметрії (у деяких сценаріях), що може створювати обмеження для ізольованих або офлайн-середовищ і вимагати додаткових заходів конфіденційності;

– складність інтеграції з застарілою інфраструктурою та потреба в кваліфікованому персоналі для налаштування й управління автоматичними реакціями та політиками.

Отже, SentinelOne представляє собою потужне рішення класу EDR/XDR, орієнтоване на проактивне виявлення й автоматичне реагування, що робить його привабливим для великих та середніх підприємств із розвинутими SOC-процедурами. Водночас досягнення заявленої ефективності потребує інвестицій у ліцензії, інтеграцію та операційну підготовку – тому перед масштабним впровадженням доцільно проводити пілотний проєкт для верифікації сумісності з наявною інфраструктурою й оцінки рівня хибних спрацьовувань.

CrowdStrike Falcon – хмарно-орієнтована платформа захисту кінцевих точок і розширеного виявлення загроз, призначена для запобігання порушенням безпеки, оперативного виявлення складних атак і прискореного реагування на

інциденти (рис. 1.6). Платформа поєднує телеметрію з кінцевих точок, глобальну розвідку про загрози та аналітику на базі ШІ для видачі миттєвих рішень щодо підозрілої активності [25].

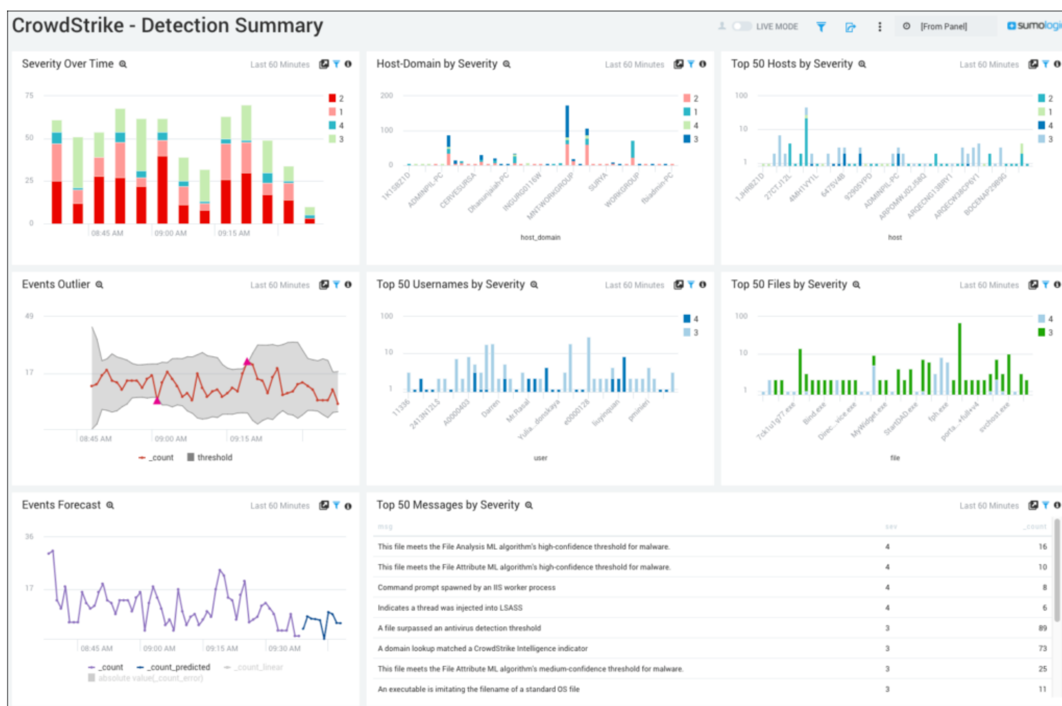

Рисунок 1.6 – Приклад інтерфейсу системи «CrowdStrike» [26]

Архітектура рішення будується навколо легкого агента (Falcon sensor), встановленого на кожній кінцевій точці, який у реальному часі збирає детальну телеметрію процесів, файлових операцій, мережевих з’єднань та системних подій і передає її у захищений хмарний сервіс; у хмарі дані індексуються й корелюються в масштабованому графовому механізмі «Threat Graph», де застосовуються моделі машинного навчання, сигнатурні й поведінкові правила, а також глобальна розвідка для формування вердиктів і генерації автоматичних дій (ізоляція вузла, блокування процесу, створення інцидентів). Платформа доповнюється сервісами керованого полювання за загрозами (OverWatch / Adversary Hunting), модулями захисту робочих навантажень у хмарі та інтеграціями з SIEM/SOC-інструментами через API, що забезпечує єдину панель управління для аналітиків і централізовану оркестрацію відповіді на інциденти.

Переваги:

– хмарно-орієнтована аналітика та глобальна розвідка загроз забезпечують швидку кореляцію телеметрії та оперативне виявлення складних ланцюжків атак;

– легкий агент на кінцевих точках з низьким накладним навантаженням дозволяє збирати детальну телеметрію без суттєвого впливу на продуктивність систем [27];

– широкий набір автоматизованих реакцій (ізоляція вузла, блокування процесу), а також сервіси керованого полювання за загрозами спрощують скорочення часу від виявлення до усунення інциденту;

– добра інтегрованість із SIEM, SOC-інструментами і можливість розширення захисту на хмарні навантаження роблять платформу придатною для гібридних інфраструктур.

Недоліками є:

– залежність від передачі телеметрії в хмару викликає питання конфіденційності, відповідності регуляторним вимогам та розміщення даних у певних юрисдикціях [28];

– відносно висока вартість ліцензування й супутніх сервісів, що може ускладнювати застосування в бюджетах малих організацій;

– потреба в кваліфікованих аналітиках і налаштуванні політик: без відповідного тюнінгу можливі надлишкові оповіщення або пропуск тонких індикаторів компрометації [29];

– часткова залежність від з’єднання з хмарними сервісами і потенційний ризик vendor-lock-in при глибокій інтеграції з екосистемою постачальника.

Отже, CrowdStrike Falcon – потужна платформа для організацій із розвинутими SOC-процедурами, що поєднує масштабовану хмарну аналітику з можливістю швидкого реагування; її застосування підвищує оперативну здатність до виявлення складних атак. Водночас ефективне використання вимагає інвестицій у ліцензування, уваги до питань приватності даних і наявності фахового персоналу для налаштування та підтримки процесів.

Bitdefender – комплексне рішення для захисту кінцевих точок і корпоративної інфраструктури, призначене для запобігання, виявлення та усунення різноманітних кіберзагроз (рис. 1.7). Основна мета платформи – забезпечити багаторівневий захист із поєднанням традиційних антивірусних механізмів та сучасних підходів (поведінковий аналіз, моделі машинного навчання, пісочниця) з можливістю централізованого моніторингу й оркестрації заходів реагування для бізнес-середовищ різного масштабу [30].

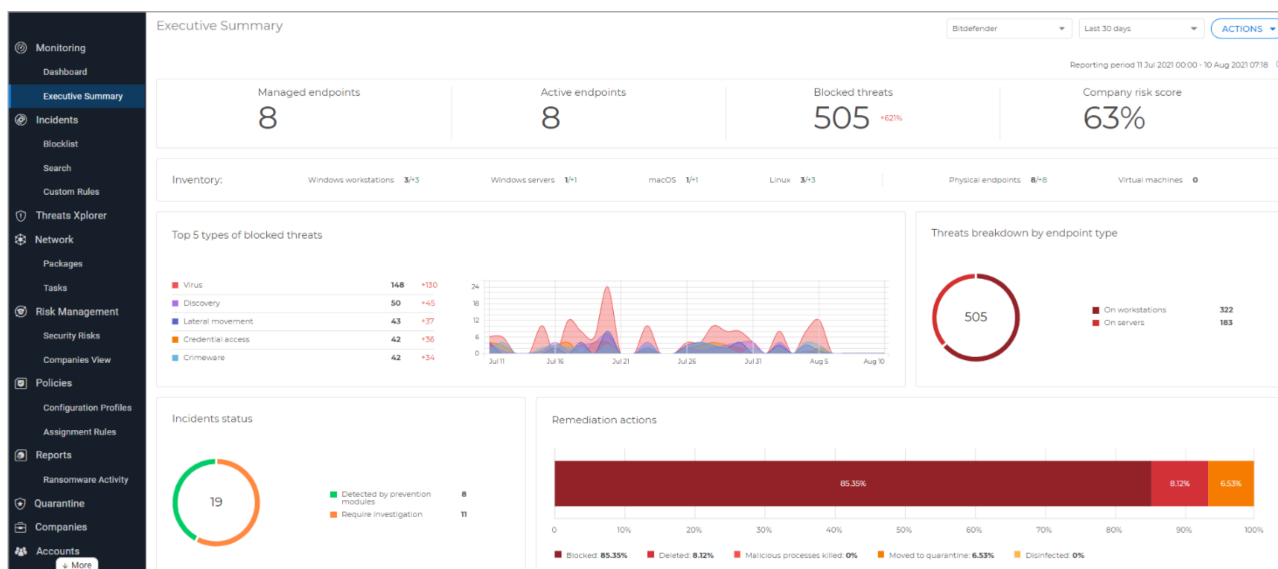

Рисунок 1.7 – Приклад інтерфейсу системи «Bitdefender» [31]

Архітектура рішення базується на тонкому агенті, встановленому на кожній кінцевій точці, який збирає локальну телеметрію та здійснює попередню фільтрацію подій, і на централізованому компоненті управління (хмарному або локальному), що виконує кореляцію, довготривалу аналітику й зберігання даних. До ключових функціональних шарів належать сигнатурний сканер та евристичні модулі, поведінковий рушій із правилами та ML-моделями для виявлення аномалій, пісочниця для динамічного аналізу підозрілих зразків і модулі відновлення для мінімізації наслідків інфікування; платформа також інтегрує оновлення репутаційних баз і телеметрію розвідки загроз для підвищення адаптивності при зміні ландшафту атак.

Переваги:

– багаторівневий механізм детекції (сигнатури, евристика, поведінковий аналіз, ML-моделі та пісочниця), що підвищує ймовірність виявлення як відомих, так і нових сімейств загроз [32];

– централізована панель управління та можливості масштабування (хмарні та гібридні розгортання), що полегшують адміністрування інструменту в корпоративному середовищі;

– засоби оперативного відновлення і ремедіації, які зменшують наслідки інцидентів та скорочують час повернення систем до працездатного стану [33];

– інтеграція з розвідувальними базами та регулярні оновлення репутаційних даних, що підвищує адаптивність системи до еволюції загроз.

Недоліки:

– потреба у тонкому налаштуванні політик і профілів для зниження кількості хибних спрацьовувань; без належного тюнінгу можливі надлишкові оповіщення [34];

– вартість ліцензій та супутніх сервісів може бути значущим фактором для невеликих організацій при масштабному розгортанні [35];

– часткова залежність від хмарних компонентів і оновлень ставить питання щодо конфіденційності телеметрії та відповідності регуляторним вимогам у деяких юрисдикціях;

– інтеграція з застарілою інфраструктурою або вузькоспеціалізованими системами може вимагати додаткових зусиль і ресурсів, особливо при глибокій кастомізації.

Отже, Bitdefender є збалансованим корпоративним рішенням для захисту кінцевих точок, яке поєднує традиційні механізми виявлення з сучасними підходами поведінкового аналізу та автоматичної ремедіації. Платформа добре підходить для організацій, що прагнуть комплексного, централізованого захисту, але для досягнення оптимальної продуктивності та мінімізації хибних тривог потребує інвестицій у налаштування, оперативну підтримку й оцінку відповідності політикам щодо обробки даних.

Аналіз існуючих систем виявлення показав, що сучасні продукти, хоча й забезпечують високий рівень автоматизації та широкі функціональні можливості, мають низку обмежень: надмірна залежність від хмарної телеметрії, висока частота хибних спрацьовувань, обмежена переносимість моделей між середовищами та вразливість до стратегій обфускації й атак на самі моделі. Через ці практичні обмеження виникає потреба в науковому опрацюванні підходів. Окрім того, відсутність єдиних протоколів валідації і репрезентативних реальних датасетів ускладнює об'єктивну оцінку й порівняння рішень, що підкреслює необхідність дослідження методів тестування та адаптації моделей у виробничих умовах. Отже, спрямоване на ці проблеми дослідження має практичну й наукову доцільність для підвищення надійності та застосовності систем протидії шкідливому ПЗ.

1.4 Постановка задачі

Проведення дослідження, присвяченого аналізу шкідливого програмного забезпечення за допомогою методів штучного інтелекту, передбачає побудову цілісної аналітичної моделі, здатної здійснювати автоматизоване розпізнавання та класифікацію загроз на основі структурованих даних про поведінку програмних об'єктів. Для цього необхідно виконати комплекс взаємопов'язаних завдань, зокрема:

– формування набору даних та препроцесінг. На цьому етапі необхідно здійснити відбір репрезентативних зразків шкідливого і легітимного програмного забезпечення, провести їхню верифікацію та очистити дані від шумових або дубльованих записів;

– розроблення методики побудови моделі. Необхідно визначити методологічні підходи до побудови класифікатора, зокрема розробити стратегії відбору інформативних ознак, способи балансування даних, а також підходи до регуляризації для запобігання перенавчанню. Важливою складовою є побудова експериментального протоколу, що визначає послідовність кроків з налаштування, навчання та валідації моделі, включно з порівнянням

альтернативних конфігурацій. Очікуваним результатом виступає набір методичних рекомендацій та конфігурацій для подальшої імплементації у програмний прототип;

– реалізація та експериментальна перевірка. На цьому етапі створюється програмний модуль, що забезпечує процес навчання та інференсу побудованої моделі. Виконується серія контрольованих експериментів із використанням крос-валідації та тестуванням на нових, раніше невідомих зразках загроз. Особлива увага приділяється аналізу чутливості до обфускації, що дозволяє оцінити здатність системи виявляти модифіковані шкідливі об’єкти. У результаті формується набір експериментальних показників – точність, повнота, F-міра, а також часові характеристики, які слугують основою для оцінювання ефективності підходу.

– оцінка стійкості та адаптивності. Після отримання попередніх результатів проводиться аналіз поведінки моделі при зміні статистичних характеристик вхідних даних, а також оцінюється її вразливість до цілеспрямованих атак. Досліджується вплив різних способів витягу ознак на стабільність класифікації, що дозволяє сформувати рекомендації щодо підтримки та оновлення моделі в умовах змінних типів шкідливого ПЗ.

Виконання поставлених завдань забезпечує створення системного підходу до аналізу шкідливих програмних об’єктів, який поєднує методи машинного навчання з експериментально перевіреними процедурами обробки та інтерпретації даних. Це дозволяє не лише підвищити рівень точності розпізнавання загроз, а й сформувати адаптивну основу для подальшого розвитку інтелектуальних систем кіберзахисту.

2 МЕТОДОЛОГІЧНІ ОСНОВИ ПОБУДОВИ МОДЕЛІ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір методу машинного навчання для виявлення ПЗ

Процес виявлення шкідливого ПЗ в сучасних інформаційних системах є складним завданням, що потребує застосування інтелектуальних методів аналізу даних. Традиційні підходи, засновані на сигнатурному порівнянні або евристичних правилах, поступово втрачають ефективність через швидку еволюцію методів маскування, поліморфізму та шифрування коду зловмисних програм. У зв’язку з цим особливої актуальності набуває використання алгоритмів машинного навчання, здатних автоматично виявляти приховані закономірності у великих обсягах даних та класифікувати ПЗ за поведінковими або статичними ознаками.

При виборі методу машинного навчання доцільно враховувати специфіку даних, що використовуються для аналізу, а також вимоги до точності, швидкодії та здатності моделі до узагальнення. У контексті задачі детектування шкідливого ПЗ застосовуються як класичні методи, так і сучасні методи машинного навчання, здатні моделювати складні нелінійні залежності між ознаками.

Метод градієнтного бустингу (Gradient Boosting) належить до ансамблевих методів машинного навчання, що базуються на принципі послідовного поєднання слабких моделей для створення однієї сильної [36]. Його основна ідея полягає у тому, щоб кожне наступне дерево рішень навчалося на залишкових похибках попередніх моделей, поступово зменшуючи помилку прогнозування. Такий підхід забезпечує високу точність і здатність виявляти складні закономірності у великих, нерівномірно збалансованих наборах даних.

Градієнтний бустинг широко використовується в аналітичних задачах класифікації, регресії та ранжування. Він є основою для сучасних високопродуктивних бібліотек, таких як XGBoost, LightGBM та CatBoost, що знайшли застосування у фінансовій аналітиці, кібербезпеці, медичних системах і зокрема – в задачах виявлення шкідливого програмного забезпечення [37]. У

цій сфері метод дозволяє ефективно класифікувати зразки ПЗ на «безпечні» та «шкідливі» на основі великої кількості поведінкових та статичних ознак, навіть якщо окремі характеристики не мають самостійної інформативної сили.

На рис. 2.1 зображено схематичну структуру процесу побудови моделі градієнтного бустингу.

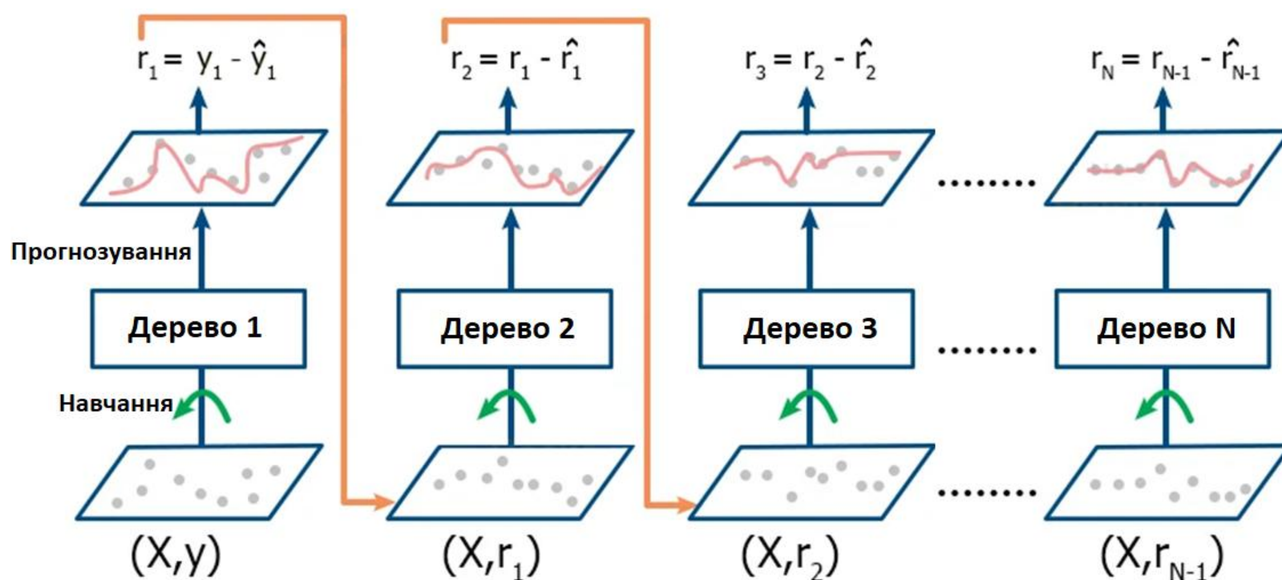

Рисунок 2.1 – Схема роботи методу градієнтного бустингу [38]

Згідно з рисунком, навчання моделі відбувається послідовно: перше дерево (дерево 1) прогнозує цільову змінну $\hat{y}_1$ на основі вхідних даних (X, y). Далі обчислюються залишки $r_1 = y - \hat{y}_1$, які відображають різницю між реальними і передбаченими значеннями. Наступне дерево (дерево 2) навчається на цих залишках, тобто на помилках попередньої моделі, з метою їх мінімізації. Процес повторюється до побудови N-го дерева, кожне з яких коригує прогноз попередніх. У результаті формується композиція дерев, що спільно наближає справжню функцію цілі, а підсумкове передбачення є зваженою сумою прогнозів усіх дерев ансамблю.

Щодо аналізу шкідливого програмного забезпечення градієнтний бустинг може застосовуватися для побудови моделей класифікації, що навчаються на великій кількості ознак, отриманих зі статичного аналізу виконуваних файлів (наприклад, структура PE-файлу, частота системних викликів, використання

мережевих функцій). Кожне дерево в ансамблі фокусується на складніших або менш очевидних патернах, які залишилися нерозпізнаними попередніми моделями. Такий підхід забезпечує високу точність і стійкість до нових, невідомих типів шкідливого ПЗ, що робить градієнтний бустинг одним із найбільш ефективних методів у сфері кіберзахисту та поведінкового аналізу програм.

Переваги методу градієнтного бустингу:

– висока точність класифікації [39]. Градієнтний бустинг здатен виявляти складні нелінійні залежності між ознаками, що робить його одним із найточніших методів серед ансамблевих підходів, особливо на нерівномірних або зашумлених даних;

– гнучкість моделювання. Алгоритм може бути адаптований до різних типів задач – класифікації, регресії, ранжування – та дозволяє використовувати широкий спектр функцій втрат і регуляризаційних стратегій;

– стійкість до змішаних типів даних. Метод ефективно працює як із числовими, так і з категоріальними або бінарними ознаками без потреби у складній попередній нормалізації чи масштабуванні даних.

Недоліки методу градієнтного бустингу:

– висока обчислювальна складність [40]. Навчання ансамблю з великої кількості дерев потребує значних ресурсів – часу, пам’яті та обчислювальної потужності, особливо при великих наборах даних;

– схильність до перенавчання. Без належної регуляризації або контролю параметрів (глибини дерев, швидкості навчання) модель може надто точно відтворювати навчальні дані, втрачаючи здатність до узагальнення;

– складність інтерпретації результатів. Через велику кількість взаємопов’язаних дерев важко пояснити, які саме ознаки вплинули на кінцеве рішення, що ускладнює використання моделі у критично важливих сферах, де потрібна пояснюваність.

Метод градієнтного бустингу є одним із найпотужніших інструментів сучасного машинного навчання, який поєднує статистичну точність із адаптивністю до різнорідних даних. Його ефективність особливо проявляється у

складних аналітичних задачах, де традиційні алгоритми не здатні забезпечити належного рівня розпізнавання. Саме тому градієнтний бустинг виступає фундаментом для побудови високопродуктивних систем кіберзахисту, аналітики поведінки та прогнозування загроз.

Метод Бройдена–Флетчера–Голдфарба–Шанно (Broyden–Fletcher–Goldfarb–Shanno, БФГШ) є одним із найефективніших алгоритмів чисельної оптимізації, який належить до класу квазі-Ньютонівських методів [41]. Його головна мета полягає у знаходженні мінімуму (або максимуму) неперервної диференційовної функції без потреби в прямому обчисленні матриці Гессе – матриці других похідних. Замість цього метод будує її апроксимацію, поступово уточнюючи наближення до точки оптимуму шляхом використання лише інформації про градієнт.

Завдяки поєднанню високої швидкості збіжності та відносної простоти реалізації, БФГШ широко використовується в задачах оптимізації параметрів моделей машинного навчання, нейронних мереж, економетричних систем, а також розпізнавання образів. Особливо ефективним цей метод є тоді, коли необхідно досягти точного мінімуму функції втрат за обмеженої кількості обчислень градієнтів.

На рис. 2.2 наведено принцип роботи методу Бройдена–Флетчера–Голдфарба–Шанно.

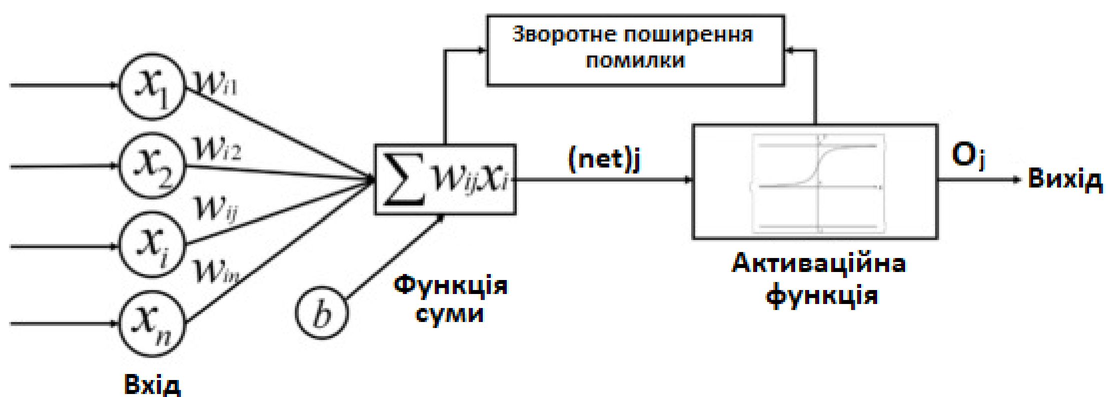

Рисунок 2.2 – Схема роботи методу Бройдена-Флетчера-Голдфарба-Шанно [42]

На вхід системи подаються ознаки x_1, x_2,…, x_n, кожна з яких має ваговий коефіцієнт w_i. У суматорі обчислюється лінійна комбінація вхідних значень де b – зміщення (поріг активації). Отримане значення передається до активаційної функції, яка визначає вихід нейрона O_j. Після обчислення результату система виконує зворотне поширення помилки, що коригує вагові коефіцієнти на основі градієнта функції втрат. Саме на цьому етапі метод БФГШ застосовується як оптимізатор, який забезпечує найшвидше зменшення помилки, визначаючи напрям і крок зміни ваг для досягнення мінімуму похибки.

Для аналізу шкідливого програмного забезпечення метод БФГШ може бути використаний для оптимізації параметрів нейронних мереж або логістичних моделей, що класифікують програми на «безпечні» та «шкідливі». Наприклад, у системах поведінкового аналізу ПЗ алгоритм дозволяє ефективно знаходити ваги, які мінімізують похибку між реальними й прогнозованими класами, базуючись на статистичних характеристиках, таких як кількість викликів API, використання пам'яті, мережеві запити тощо.

Переваги методу БФГШ:

– швидка збіжність [43]. Завдяки апроксимації матриці Гессе метод забезпечує квадратичну або наближену до квадратичної швидкість збіжності, що дозволяє досягати мінімуму функції втрат за меншу кількість ітерацій порівняно з градієнтними методами першого порядку;

– висока точність. Алгоритм ефективно враховує геометрію функції втрат, коригуючи напрямок пошуку не лише за градієнтом, а й за змінами у кривині функції, що сприяє більш точному знаходженню локального мінімуму;

– стійкість до поганих початкових умов [44]. Завдяки оновленню оберненої матриці Гессе на основі попередніх градієнтів метод демонструє стабільну роботу навіть при невдалому виборі початкових вагових коефіцієнтів або параметрів.

Недоліки методу Бройдена–Флетчера–Голдфарба–Шанно:

– високі обчислювальні витрати. Обчислення та збереження апроксимованої матриці Гессе вимагає значних ресурсів, особливо при великій

кількості параметрів, що робить метод менш придатним для глибоких нейронних мереж із мільйонами ваг;

– чутливість до розмірності даних. Для задач із великою кількістю ознак метод може втрачати ефективність через збільшення складності оновлення матриці наближення;

– схильність до локальних мінімумів [45]. Як і більшість детермінованих методів оптимізації, БФГШ не гарантує знаходження глобального мінімуму, особливо у складних, багатовимірних ландшафтах функцій втрат.

Отже, метод БФГШ поєднує математичну точність із практичною ефективністю, дозволяючи точно налаштовувати параметри моделей навіть за складних умов. Його використання забезпечує підвищення аналітичної точності систем, що працюють із великими наборами поведінкових або структурних характеристик, а тому він є дієвим інструментом у дослідженнях, спрямованих на вдосконалення алгоритмів виявлення шкідливого програмного забезпечення.

Метод логістичної регресії належить до базових, але надзвичайно ефективних алгоритмів машинного навчання, що використовуються для бінарної класифікації [46]. Його головна ідея полягає у встановленні залежності між набором незалежних змінних (ознак) та цільовою змінною, яка може приймати лише два стани – наприклад, «1» для шкідливого програмного забезпечення та «0» для безпечного. На відміну від лінійної регресії, логістична регресія не прогнозує безпосереднє числове значення, а обчислює ймовірність належності об’єкта до певного класу.

Математично метод базується на використанні логістичної (сигмоїдної) функції активації, що обмежує вихідні значення у діапазоні [0;1]. Це дозволяє інтерпретувати результат моделі як ймовірність. Завдяки своїй стабільності, простоті реалізації та інтерпретованості, логістична регресія є популярним інструментом у таких сферах, як медична діагностика, аналіз ризиків, виявлення шахрайства, кібербезпека та розпізнавання шкідливого ПЗ.

На рис. 2.3 подано узагальнену схему функціонування методу логістичної регресії.

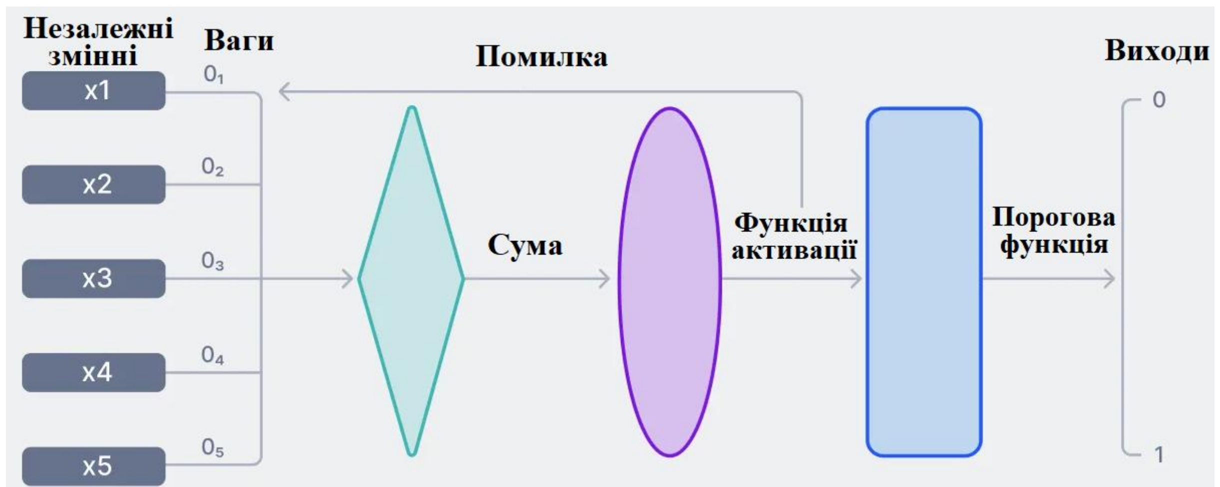

Рисунок 2.3 – Схема роботи методу логістичної регресії [38]

На вхід моделі подається набір незалежних змінних $x_1, x_2, \ldots, x_5$, які характеризують об’єкт аналізу. Кожній ознаці відповідає певна вага w_i, що визначає ступінь її впливу на результат. Далі всі вхідні дані підсумовуються за допомогою функції суми, утворюючи зважене значення:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + \ + w_n x_n. \quad (2.1)$$

Після цього результат передається через логістичну функцію активації, яка нормалізує значення у межах від 0 до 1:

$$P(y = 1 \mid x) = \frac{1}{1+e^{-z}}. \quad (2.2)$$

Отримана ймовірність далі проходить через порогову функцію, що перетворює її у кінцевий класовий результат – «1» або «0». У процесі навчання модель коригує вагові коефіцієнти шляхом мінімізації помилки, використовуючи, наприклад, алгоритм градієнтного спуску.

У контексті аналізу шкідливого програмного забезпечення, логістична регресія може застосовуватися для класифікації виконуваних файлів або процесів на основі їхніх характеристик – наприклад, кількості мережевих викликів, використання пам’яті, системних функцій, структури коду чи частоти звернень до ресурсів. Модель обчислює ймовірність того, що конкретна програма є шкідливою, і на основі встановленого порогу приймає рішення про її категорію.

Переваги методу логістичної регресії:

– інтерпретованість результатів. Логістична регресія забезпечує чітке математичне тлумачення вагових коефіцієнтів, що дозволяє аналітику оцінити вплив кожної змінної на ймовірність належності до певного класу;

– швидкість і простота реалізації [48]. Метод має низьку обчислювальну складність і швидко навчається навіть на великих наборах даних, що робить його зручним для побудови базових або реальних оперативних систем класифікації;

– стійкість до шумових даних. Завдяки використанню ймовірнісної інтерпретації модель демонструє стабільні результати навіть у присутності частково некоректних або зашумлених спостережень.

Недоліки методу логістичної регресії:

– нелінійні залежності. Модель ефективно працює лише у випадках, коли зв’язок між змінними має приблизно лінійну природу; складні або багатовимірні нелінійності вона описує з обмеженою точністю;

– чутливість до багатоколінеарності [49]. Якщо незалежні змінні сильно корельовані між собою, точність і стабільність оцінки вагових коефіцієнтів можуть суттєво знижуватись;

– неадаптивність до великої кількості ознак. При надмірній кількості змінних або складних структурованих даних (наприклад, у задачах з великим обсягом поведінкових параметрів) модель може втратити ефективність без попереднього відбору ознак або регуляризації.

Логістична регресія є фундаментальним статистичним інструментом, який поєднує простоту з аналітичною точністю, формуючи основу для складніших методів машинного навчання. Її застосування в системах аналізу шкідливого програмного забезпечення дозволяє створювати базові предиктивні моделі, що забезпечують швидку та обґрунтовану оцінку ризику навіть за обмеженої кількості навчальних даних.

Для узагальнення отриманих результатів і кращого розуміння особливостей розглянутих методів доцільно подати їх порівняння у табличній формі. Це дозволяє простежити ключові відмінності між підходами за основними технічними та аналітичними критеріями, що визначають їхню

придатність до вирішення завдань класифікації у сфері аналізу шкідливого програмного забезпечення.

Таблиця 2.1.

Порівняльна характеристика методів машинного навчання

Характеристика	Градієнтний бустинг	БФГШ	Логістична регресія
Тип моделі	Ансамблевий	Оптимізаційний	Лінійна ймовірнісна модель
Швидкість збіжності	Помірна, залежить від кількості дерев	Висока, завдяки апроксимації матриці Гессе	Висока, але знижується на великих даних
Точність на складних даних	Висока за рахунок ансамблю	Висока при оптимізації багатопараметричних моделей	Середня, переважно для лінійних залежностей
Вимоги до обчислювальних ресурсів	Значні (через послідовне навчання дерев)	Помірні, стабільні для середніх розмірів даних	Низькі
Стійкість до зашумлених даних	Добра, але залежить від глибини дерев	Висока завдяки стабільній корекції ваг	Помірна, чутлива до вибірки
Інтерпретованість результатів	Складна (через ансамблеву структуру)	Зберігається можливість аналітичного аналізу	Висока, ваги легко тлумачити
Можливість оптимізації функції втрат	Залежить від типу задачі	Гнучка, підлаштовується під різні функції	Обмежена до логістичної функції
Застосування у класифікації ПЗ	Виявлення складних поведінкових закономірностей	Оптимізація моделей для точного навчання класифікаторів	Базова оцінка ймовірності належності ПЗ до класу

Вибір методу БФГШ для розробки системи зумовлений його здатністю забезпечувати швидку збіжність під час оптимізації параметрів моделей, що

особливо важливо для аналітичних систем у режимі реального часу. Завдяки використанню апроксимації матриці Гессе метод дозволяє досягати високої точності без суттєвих витрат обчислювальних ресурсів. Його стійкість до зашумлених даних і можливість адаптації до різних функцій втрат роблять його ефективним для задач класифікації, пов’язаних із виявленням шкідливого програмного забезпечення. Крім того, БФГШ забезпечує аналітичну інтерпретованість процесу оптимізації, що сприяє підвищенню надійності та пояснюваності результатів у контексті кіберзахисних систем.

2.2 Вибір та опис набору даних для навчання і тестування моделі

Для проведення експериментальних досліджень було використано відкритий набір даних “Classification based PE dataset on benign and malware files”, розміщений на платформі Kaggle [50]. Даний набір містить збалансовану вибірку з 100 000 зразків виконуваних файлів, серед яких 50 000 належать до шкідливого програмного забезпечення, а 50 000 – до безпечних програм. Така структура забезпечує репрезентативність і дозволяє уникнути зміщення моделі внаслідок дисбалансу класів. Джерело даних обране завдяки його відкритості, технічній достовірності та широкому використанню в дослідницьких роботах, присвячених проблемам аналізу шкідливого коду.

Набір даних сформовано на основі характеристик Portable Executable (PE) файлів, отриманих за допомогою спеціалізованої бібліотеки Python. Кожен запис містить унікальний хеш-файл, часові та системні параметри виконання, а також велику кількість поведінкових і структурних ознак, що відображають внутрішню активність програмного процесу. Зокрема, враховуються такі характеристики, як кількість звернень до пам’яті, використання пріоритетів виконання, обсяг віртуального простору, частота викликів системних функцій, кількість сторінок пам’яті, а також параметри користувацької активності під час виконання.

Перед початком моделювання проведено етап попередньої обробки, який включав очищення від дубльованих записів, перевірку цілісності хешів, нормалізацію числових параметрів та перетворення категоріальних ознак у

числовий формат. Після цього дані було розділено на навчальну (70%) та тестову (30%) вибірки. Навчальна вибірка використовувалась для побудови моделі з оптимізацією вагових коефіцієнтів за методом Бройдена–Флетчера–Голдфарба–Шанно, тоді як тестова – для незалежної перевірки точності класифікації та оцінки узагальнюючої здатності моделі.

У табл. 2.2 наведено опис основних атрибутів тренувального набору, які використовувалися при формуванні моделі для виявлення шкідливого програмного забезпечення. Атрибути містять як системні, так і поведінкові параметри, що мають безпосередній вплив на процес класифікації.

Таблиця 2.2.

Опис атрибутів тренувального набору для аналізу шкідливого ПЗ

№	Назва атрибуту	Тип даних	Опис
1	2	3	4
1	hash	string	Унікальний хеш-файл, що ідентифікує програму
2	millisecond	integer	Час виконання процесу у мілісекундах
3	classification	string	Клас об’єкта: malware або benign
4	usage_counter	integer	Лічильник використання ресурсу під час виконання
5	prio / static_prio / normal_prio	integer	Параметри пріоритетності процесу
6	policy	integer	Політика планування виконання процесу
7	task_size	integer	Обсяг пам’яті, виділений процесу (у байтах)
8	map_count	integer	Кількість відображених сегментів пам’яті
9	total_vm / shared_vm / exec_vm	integer	Характеристики віртуальної пам’яті процесу
10	utime / stime / gtime	float	Час користувацького, системного та глобального режимів виконання
11	signal_nvcsw / nvcsw / nivcsw	integer	Кількість контекстних перемикань у процесі
12	min_flt / maj_flt	integer	Кількість незначних і значних помилок сторінки

13	lock	integer	Показник блокування ресурсу системою
14	state	integer	Поточний стан процесу під час виконання
15	vm_pgoff	integer	Зміщення сторінки у віртуальній пам’яті

Продовження таблиці 2.2.

1	2	3	4
16	vm_truncate_count	integer	Кількість скорочень або змін розміру віртуального простору процесу
17	cached_hole_size	integer	Розмір кешованих вільних ділянок у пам’яті (байти), показує ефективність використання ресурсів
18	free_area_cache	integer	Обсяг вільного простору в пам’яті процесу, доступний для розміщення нових сегментів
19	hiwater_rss	integer	Максимальне значення споживання фізичної пам’яті під час виконання процесу
20	reserved_vm	integer	Обсяг віртуальної пам’яті, зарезервованої системою під час роботи процесу
21	nr_ptes	integer	Кількість таблиць сторінок (Page Table Entries), що характеризує складність структури пам’яті
22	end_data	integer	Адреса завершення секції даних у пам’яті виконуваного файлу
23	last_interval	float	Інтервал часу між останніми подіями процесу, що характеризує активність виконання
24	fs_excl_counter	integer	Лічильник виключного доступу до файлової системи (операції блокування/запису)
25	cgtime	float	Час, витрачений процесом у груповому контексті
26	gtime	float	Глобальний час обробки, який охоплює роботу процесу у всіх режимах виконання
27	signal_nvcsw	integer	Кількість контекстних перемикань, спричинених сигналами системи або користувача
28	maj_flt	integer	Кількість значних сторінкових помилок (page faults), що вимагають звернення до диску
29	min_flt	integer	Кількість незначних сторінкових помилок, що не потребують завантаження даних із диску

30	nvcsw / nivcsw	integer	Кількість добровільних і недобровільних перемикань контексту

Для більш глибокого розуміння взаємозв’язків між атрибутами, наведеними у табл. 2.2, доцільно здійснити кореляційний аналіз. Такий підхід дозволяє визначити ступінь взаємної залежності між параметрами, виявити надлишкові або дубльовані ознаки, а також зменшити розмірність даних без втрати інформативності. Результати цього аналізу представлено у вигляді теплової карти на рис. 2.4, де відображено кореляційну матрицю коефіцієнтів Пірсона для числових змінних навчального набору даних.

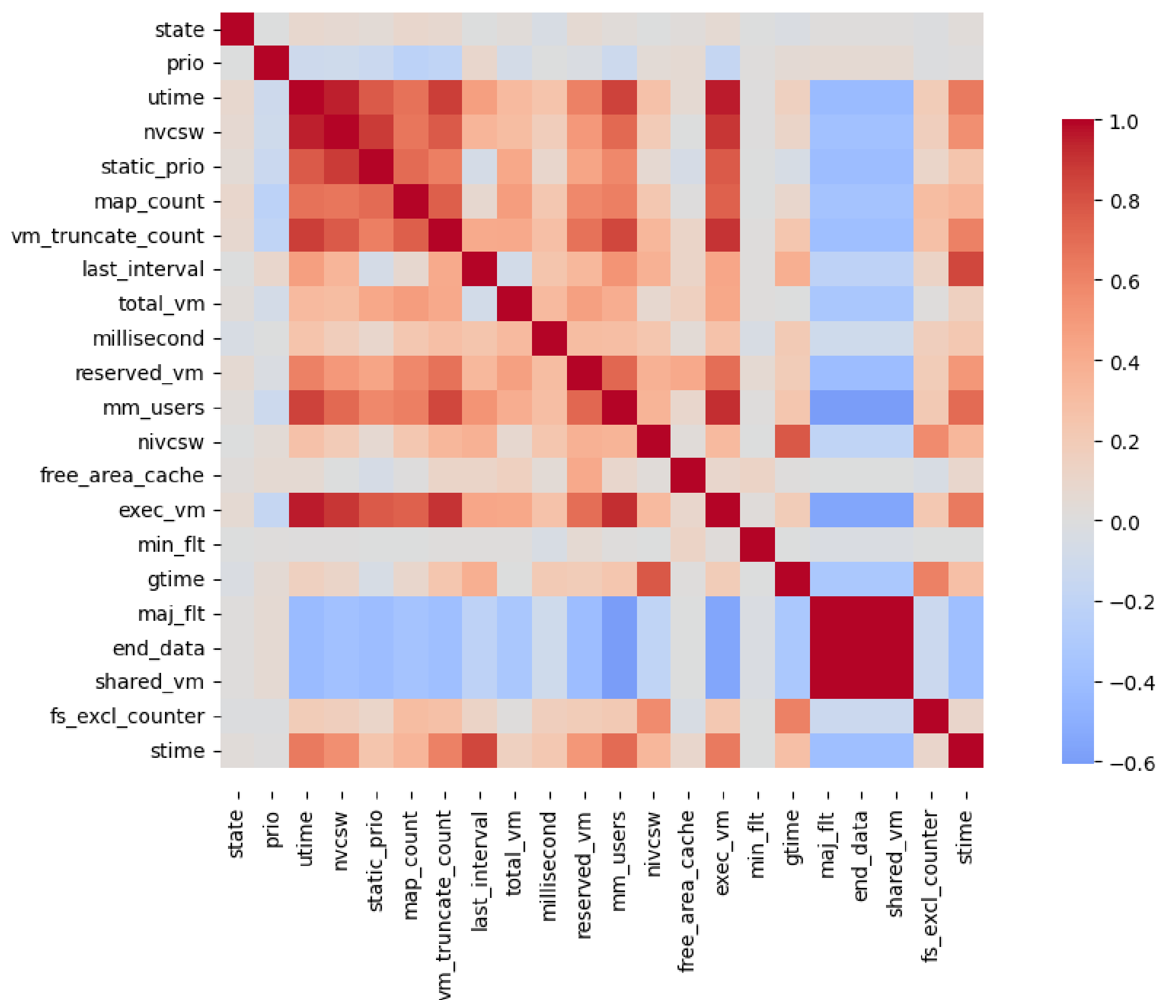

Рисунок 2.4 – Теплова карта кореляції Пірсона між атрибутами вибірки

Найбільш тісні кореляційні зв’язки спостерігаються між параметрами prio, static_prio та normal_prio, що свідчить про їхню подібну поведінку в описі пріоритетності процесів. Також виявлено значну залежність між показниками total_vm, reserved_vm і exec_vm, що логічно пояснюється їх належністю до групи характеристик управління віртуальною пам’яттю. Високий рівень кореляції між цими змінними може свідчити про надлишковість інформації, тому в процесі підготовки даних доцільно залишати лише одну з них для запобігання мультиколінеарності.

У той же час низька або від’ємна кореляція між такими параметрами, як maj_flt, nivcsw, free_area_cache та stime, вказує на їх незалежну природу та потенційно високу інформативність для класифікації поведінкових відмінностей між шкідливими та безпечними програмами. Це дозволяє моделі ефективніше виявляти аномальні процеси, пов’язані з надмірним споживанням пам’яті, збоями у сторінкових таблицях чи нехарактерною динамікою виконання. Таким чином, проведений аналіз підтверджує доцільність використання кореляційної фільтрації під час відбору ознак, що забезпечує підвищення узагальнюючої здатності та точності побудованої моделі.

На рис. 2.5 подано приклад тривимірної проєкції даних за трьома ключовими атрибутами: state, prio та utime.

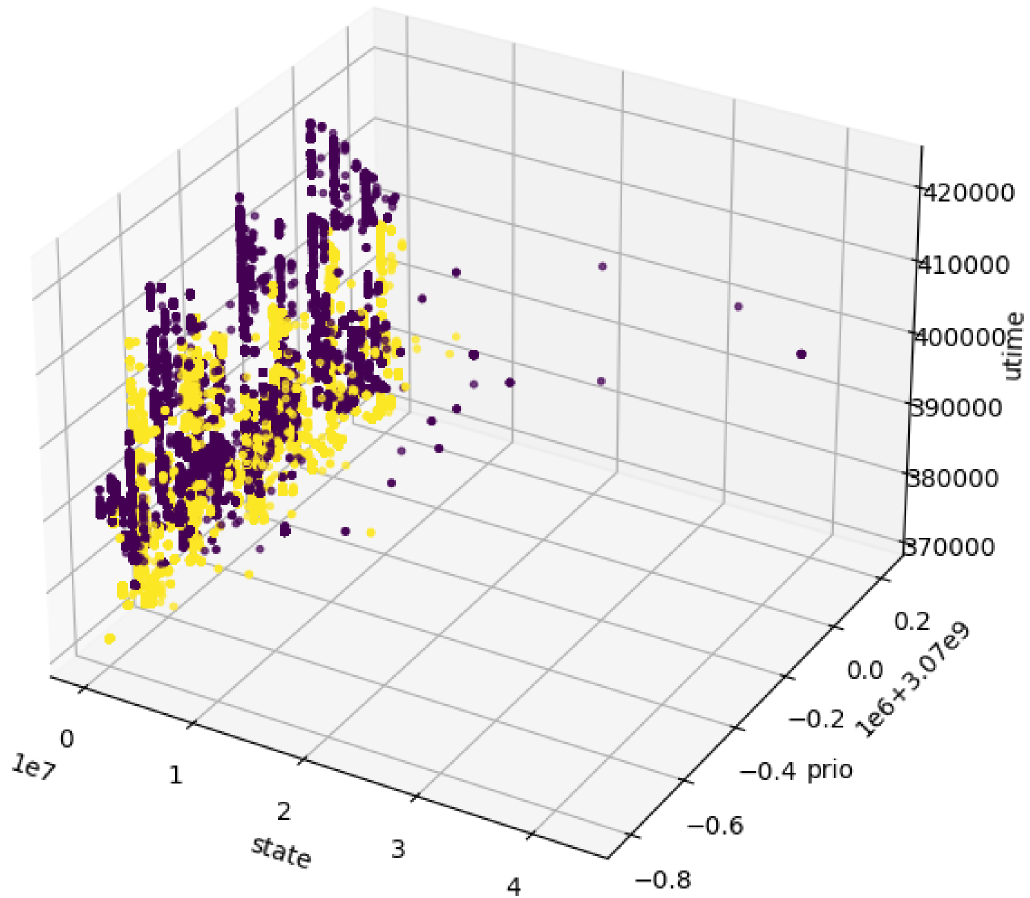

Рисунок 2.5 – 3D-розподіл атрибутів навчального набору даних

Як видно з рисунку, точки, що належать до різних класів (жовті – безпечні, фіолетові – шкідливі), утворюють групи з певними областями скупчення, які відрізняються за щільністю та просторовим положенням. Параметр state, який описує стан процесу під час виконання, демонструє помітну варіативність у розподілі між двома класами. У свою чергу, prio (пріоритет виконання процесу) та utime (час користувацького режиму) відображають відмінності у рівнях використання системних ресурсів – шкідливі процеси зазвичай характеризуються більшими коливаннями значень, що свідчить про їх агресивнішу поведінку.

Для оцінювання можливості візуального розділення двох класів даних – шкідливого та безпечного програмного забезпечення – було виконано двовимірне відображення простору ознак за допомогою методу UMAP (Uniform Manifold Approximation and Projection). На рис. 2.6 наведено результат візуалізації даних за допомогою UMAP, де точки різних кольорів відображають два класи – malware (сині точки) та benign (помаранчеві точки).

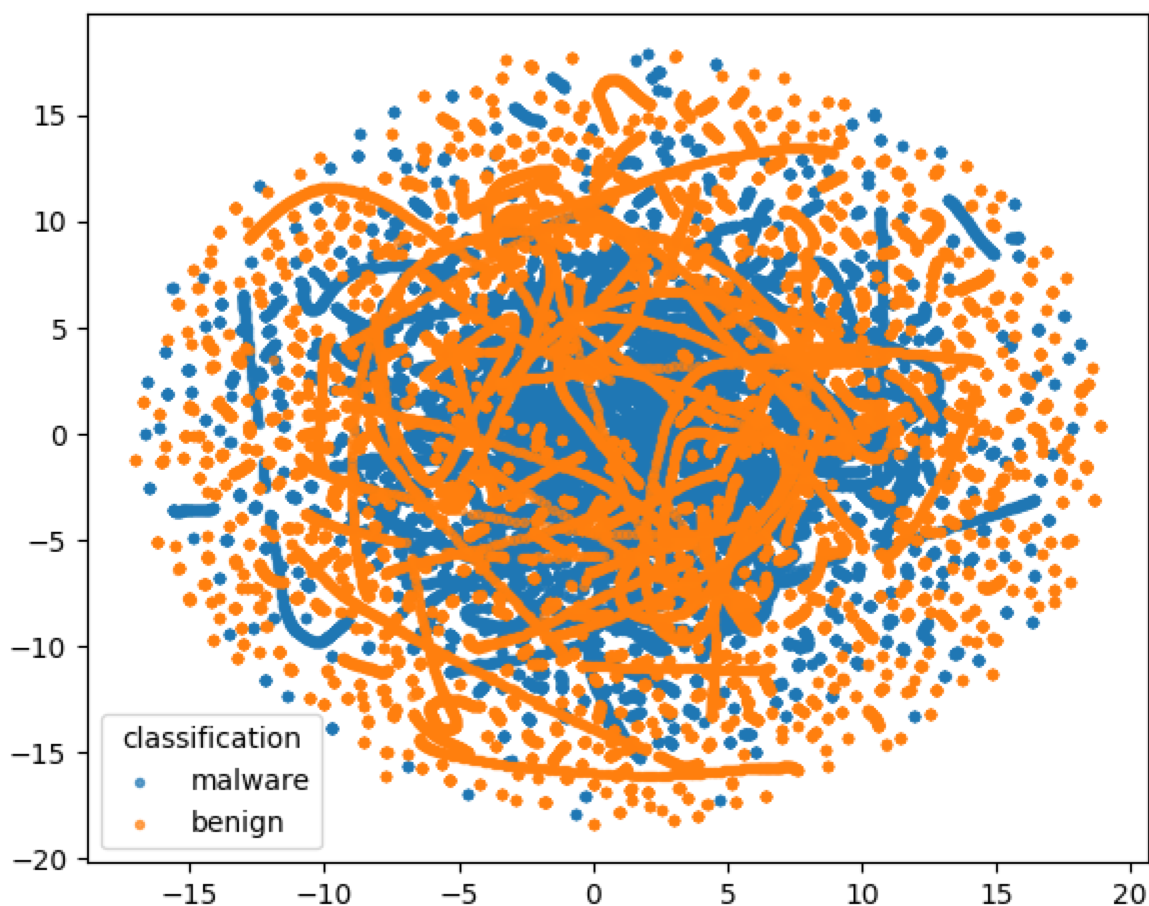

Рисунок 2.6 – UMAP 2D-проєкція

Розподіл об’єктів утворює помітні зони концентрації, що вказує на наявність кластерної структури у вибірці. Хоча області двох класів частково перекриваються, спостерігається тенденція до формування груп із відмінною щільністю, особливо у центральній та периферійній частинах проєкції. Це свідчить про наявність латентних відмінностей у поведінкових характеристиках шкідливих і безпечних процесів, які можуть бути використані для побудови моделі класифікації.

Застосування методу UMAP дозволило підтвердити, що навіть за великої кількості атрибутів дані зберігають виражену структуру, придатну для подальшого навчання алгоритмів машинного навчання. Такий результат демонструє якість сформованого датасету та обґрунтовує доцільність використання методів оптимізації (зокрема, БФГШ) для побудови точної моделі виявлення шкідливого програмного забезпечення.

Для оцінки розподілу окремих числових ознак між двома класами – malware та benign – було виконано побудову KDE-графіка (Kernel Density Estimation), який відображає неперервну апроксимацію щільності ймовірності. На рис. 2.7 наведено результат аналізу атрибуту map_count, який характеризує кількість відображених сегментів пам’яті, що використовуються процесом під час виконання.

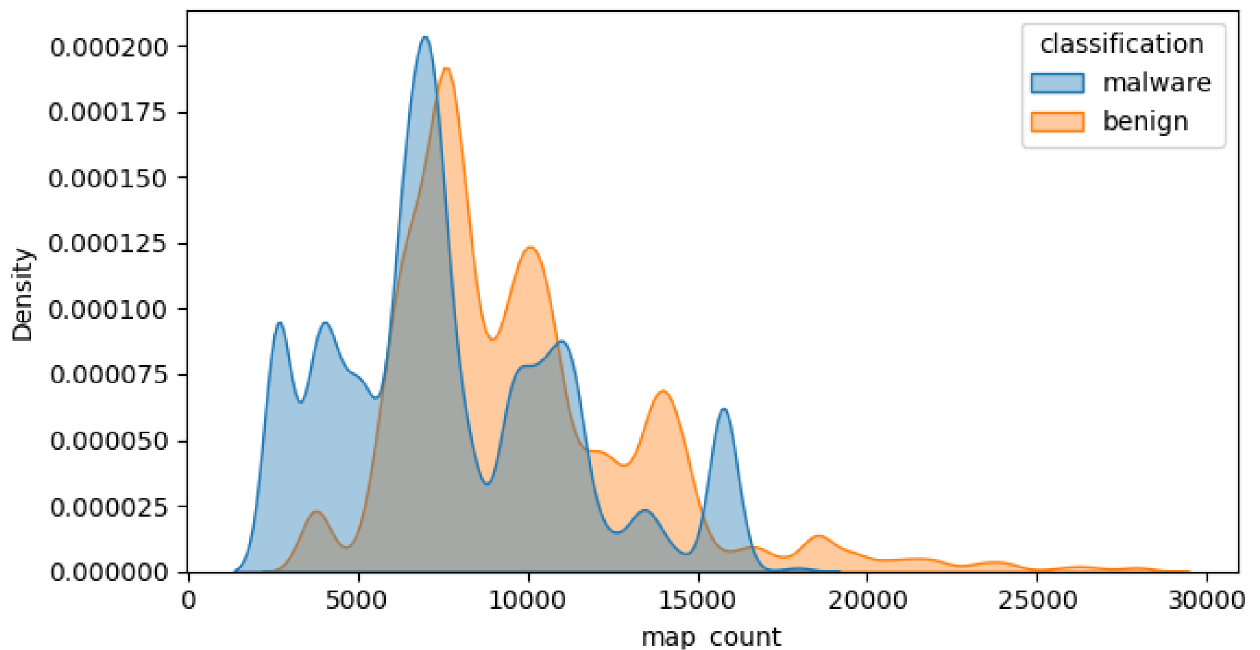

Рисунок 2.7 – KDE-розподіл map_count за класифікацією

Як видно з рисунка, обидва класи мають багатомодальний розподіл із кількома локальними максимумами, однак щільність розподілу для шкідливих програм (malware, синя лінія) зосереджена переважно в межах нижчих значень map_count, тоді як безпечні процеси (benign, помаранчева лінія) демонструють більшу варіативність і зміщення щільності у бік вищих значень. Це свідчить про те, що шкідливе ПЗ частіше використовує обмежену кількість сегментів пам’яті, зберігаючи стабільну поведінкову структуру для уникнення виявлення, тоді як легітимні програми відзначаються ширшим діапазоном звернень до пам’яті внаслідок більшої складності виконуваних функцій.

Отримані результати підтверджують, що атрибут map_count має достатню аналітичну значущість для подальшого включення до моделі класифікації, оскільки він демонструє різницю у поведінкових патернах між двома групами.

Такий підхід до візуального аналізу розподілів дозволяє не лише оцінити інформативність ознак, а й підвищити інтерпретованість майбутніх результатів машинного навчання у системі виявлення шкідливого програмного забезпечення.

Для поглибленого розуміння взаємозв'язків між основними параметрами датасету було виконано побудову мережевої візуалізації кореляційних зв'язків, що дозволяє ідентифікувати групи ознак, які демонструють високий ступінь взаємної залежності. Такий підхід дає змогу не лише оцінити структурну організацію даних, але й виявити можливі надлишкові параметри, які можуть бути виключені з подальшого аналізу без втрати інформативності моделі. На рис. 2.8 представлено граф кореляцій, сформований на основі значень коефіцієнта Пірсона вище 0.7.

Найбільш щільно пов'язаними між собою є атрибути utime, nvcsw, exec_vm, vm_truncate_count та map_count, що формують компактний кластер, який відображає спільні характеристики, пов'язані з інтенсивністю використання ресурсів процесором та пам'яттю. Зокрема, сильна кореляція між utime та exec_vm (0.96) свідчить про закономірність: збільшення часу виконання користувацького коду супроводжується розширенням обсягу віртуальної пам'яті, задіяної процесом. Аналогічно, тісний зв'язок між map_count і vm_truncate_count вказує на спільні механізми керування сторінками пам'яті.

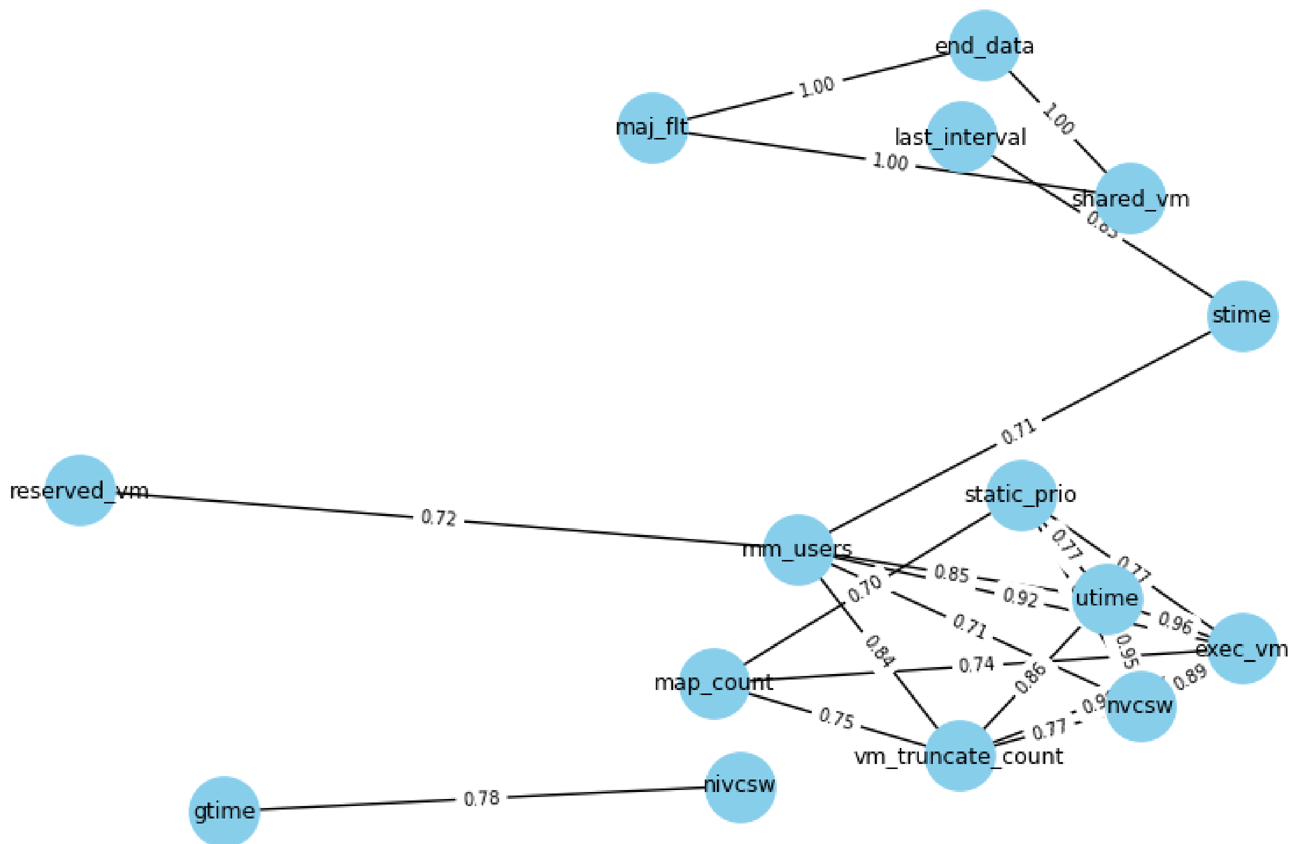

Рисунок 2.8 – Мережева візуалізація сильних кореляцій між атрибутами навчальної вибірки

Окрему групу складають показники stime, shared_vm, last_interval і maj_flt, які характеризують системні аспекти виконання процесів – обробку переривань, спільний доступ до пам’яті та взаємодію із файловими ресурсами. Водночас більш ізольованими у структурі графа залишаються reserved_vm та gtime, що відображає їх відносну незалежність і потенційну інформативність для розпізнавання поведінкових аномалій.

Отримана мережева структура дозволяє виявити основні групи взаємопов’язаних змінних, оптимізувати набір ознак для подальшого навчання моделі та зменшити ризик мультиколінеарності. Такий підхід підвищує стабільність алгоритму класифікації та сприяє більш точному відокремленню шкідливих програмних процесів від безпечних у системі виявлення шкідливого програмного забезпечення.

Для визначення внеску кожної ознаки у процес класифікації було проведено оцінювання важливостей атрибутів (feature importance), що дозволяє

кількісно оцінити ступінь впливу кожного параметра на кінцеве рішення моделі. Такий аналіз є ключовим етапом у побудові системи виявлення шкідливого програмного забезпечення, оскільки він дає змогу виокремити найбільш інформативні характеристики, скоротити розмірність даних і підвищити інтерпретованість результатів. На рис. 2.9 представлено теплову карту розподілу вагомостей ознак, отриману в результаті побудови моделі.

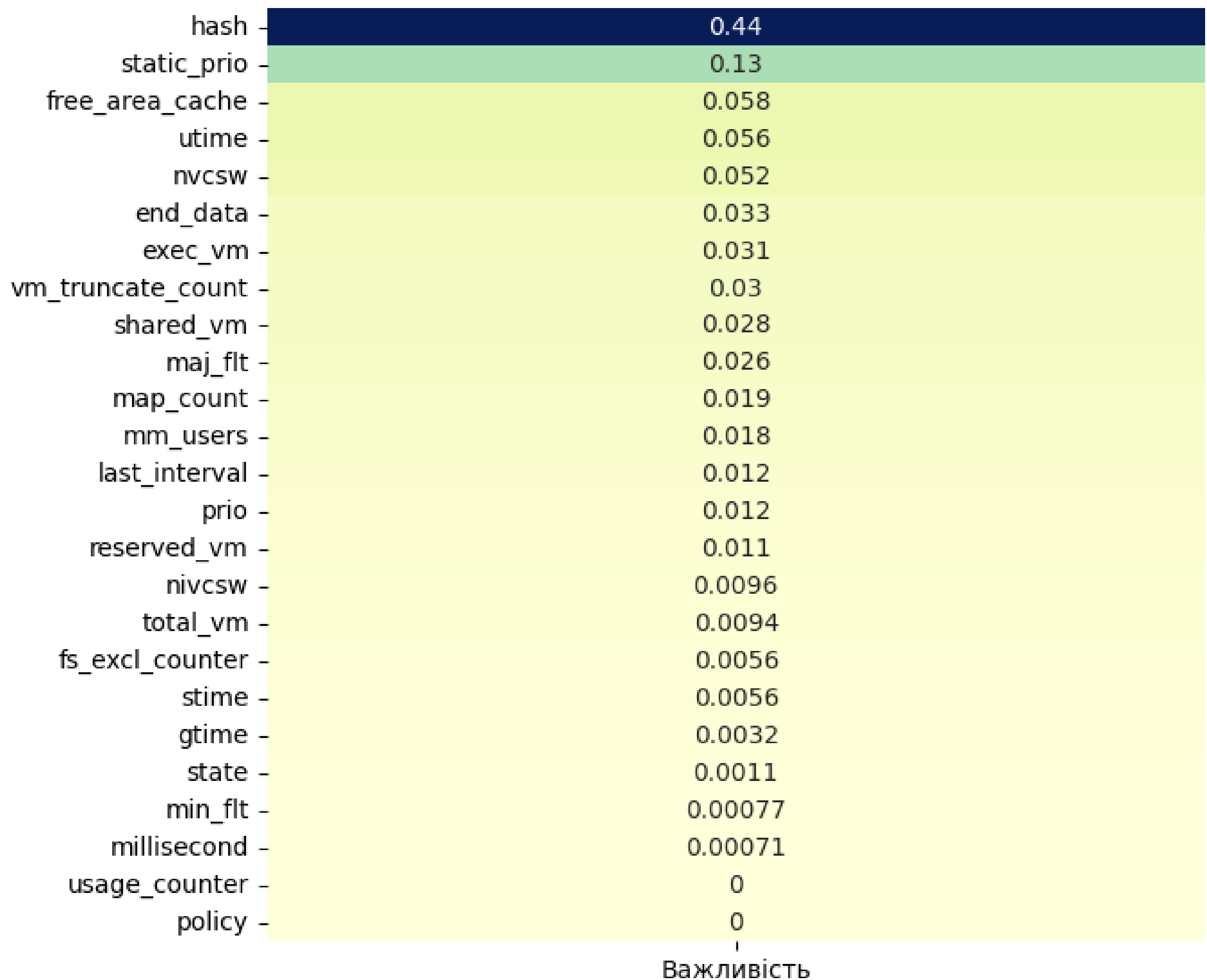

Рисунок 2.9 – Теплова карта важливостей ознак

Як видно з рисунка, найбільший вплив на якість класифікації має атрибут hash (0.44), який виступає унікальним ідентифікатором виконуваного файлу та містить непрямі закономірності, пов’язані з характером формування бінарних структур шкідливих програм. Значний внесок також спостерігається у параметра static_prio (0.13), що визначає статичний пріоритет процесу в операційній

системі. Високе значення цієї ознаки може бути пов’язане з тим, що шкідливі програми часто змінюють власний пріоритет для забезпечення прихованого або привілейованого виконання.

Серед помітних атрибутів також виділяються free_area_cache, utime, nvcsw та end_data, які відображають особливості використання пам’яті та поведінку процесу при виконанні. Їхня сукупна важливість свідчить про те, що комбінація поведінкових та ресурсних характеристик є ключовим чинником для виявлення аномалій у роботі програм. Натомість такі параметри, як usage_counter, policy та millisecond, мають мінімальну інформативність і можуть бути виключені з фінальної моделі без суттєвої втрати точності.

Таким чином, аналіз важливостей ознак підтверджує, що ефективна класифікація шкідливого ПЗ базується насамперед на характеристиках, пов’язаних із внутрішніми параметрами процесів, управлінням пріоритетами та структурою пам’яті. Отримані результати дозволили скоригувати набір вхідних змінних, підвищивши точність і стабільність моделі машинного навчання при подальшому її тестуванні.

2.3 Розроблення алгоритму машинного навчання для класифікації ПЗ

Розроблення алгоритму машинного навчання для класифікації програмного забезпечення передбачає побудову цілісного процесу, що охоплює підготовку даних, формування простору ознак, вибір моделі та її параметризацію, а також подальше навчання з контролем узагальнювальної здатності.

Кожен запис у вхідному наборі даних – це сукупність числових характеристик процесу виконання програми, таких як *state*, *usage_counter*, *vm_pgoff*, *nvcsw*, *signal_nvcsw* тощо. Ці атрибути формують простір ознак, у якому відбувається аналіз:

$$x = (x_1, x_2, \dots, x_d) \in R^d, \quad (2.3)$$

де d – кількість незалежних параметрів, що описують поведінку процесу.

Усі вектори x об'єднуються у матрицю ознак $X \in R^{n \times d}$, де n – кількість спостережень (процесів або програм).

Модель має вирішити задачу бінарної класифікації, де кожному запису присвоюється мітка:

$$y_i = \begin{cases} 1, \text{якщо зразок є шкідливим } (malware), \\ 0, \text{якщо зразок є безпечним } (benign). \end{cases} \quad (2.4)$$

У програмному коді це реалізовано через перетворення текстового значення classification у булеву змінну *Label*. Така дискретна природа змінної yi дозволяє навчати класифікатор на основі логістичної функції втрат, що інтерпретує ймовірність належності до позитивного класу.

Перед навчанням виконується мін–макс нормування, що масштабує всі ознаки до інтервалу [0, 1]. Це потрібно для стабільності обчислень під час оптимізації БФГШ, адже різні масштаби вхідних величин можуть викликати числові викривлення. Формально:

$$x'_{ij} = \frac{x_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)}, \forall \mathrm{i} = 1..\mathrm{n}, \mathrm{j} = 1..\mathrm{d}. \quad (2.5)$$

Результатом цього кроку є рівноважний внесок кожної ознаки у процес побудови моделі, що запобігає домінуванню атрибутів із великими числовими значеннями (наприклад, час виконання у мілісекундах) над менш масштабними параметрами (кількість перемикань контексту тощо). Для кожного запису після нормування ознак отримується:

$$p_\theta(y_i = 1 | x'_i) = \frac{1}{1 + e^{-\left(w^T x'_i + b\right)}}, \quad (2.6)$$

де:

$w = w_1, w_2, \dots, w_d$ – вектор вагових коефіцієнтів моделі;

b – зміщення (bias);

$\theta = (w, b)$ – сукупність параметрів, що підлягають оптимізації.

Значення $p_\theta(y_i = 1 | x'_i)$ інтерпретується як ступінь впевненості у тому, що конкретна програма є шкідливою. Це формулювання має гладку логістичну форму, що дозволяє застосовувати методи градієнтної оптимізації.

Після обчислення ймовірності належності до класу «malware» застосовується порогове правило для прийняття рішення. Якщо отримана

ймовірність перевищує заданий поріг τ (типово 0.5), система класифікує зразок як шкідливий:

$$\hat{y}_i = \begin{cases} 1, \text{якщо } p_\theta(y_i = 1|x_i') \geq \tau, \\ 0, \text{якщо } p_\theta(y_i = 1|x_i') < \tau. \end{cases} \quad (2.7)$$

Параметр τ може регулювати компроміс між чутливістю (виявленням шкідливих об'єктів) та специфічністю (уникненням хибних спрацьовувань).

Після того як модель визначена, її параметри $\theta=(w,b)$ мають бути обрані так, щоб імовірності $p_\theta(y_i = 1|x_i')$ максимально відповідали фактичним класам y_i. Це досягається шляхом мінімізації логарифмічної функції втрат, яка вимірює невідповідність прогнозованих ймовірностей істинним значенням:

$$L(w, b; x_i', y_i) = -[y_i \ln p_\theta(y_i = 1|x_i') + (1 - y_i) \ln(1 - p_\theta(y_i = 1|x_i'))], \quad (2.8)$$

Ця функція має властивість строгої опуклості, що гарантує існування єдиного глобального мінімуму. Інтуїтивно вона карає як надмірно впевнені неправильні передбачення, так і надто невпевнені правильні прогнози, стимулюючи модель видавати добре калібровані ймовірності.

Для всього тренувального набору даних з n прикладів обчислюється середня (емпірична) втрата:

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} l\,(w, b; x_i', y_i). \quad (2.9)$$

Це усереднення забезпечує рівноправний вплив кожного зразка на загальну функцію втрат, що особливо важливо при нерівномірному розподілі класів у вибірці.

Для запобігання переобладнанню (overfitting) додають член регуляризації, який контролює величину вагових коефіцієнтів. У нашому випадку використовується $L2$-регуляризація, тобто штраф за велику норму вектора ваг:

$$\Omega(w) = \frac{\lambda}{2} \|w\|_2^2 = \frac{\lambda}{2} \sum_{j=1}^{d} w_j^2, \quad (2.10)$$

де $\lambda>0$ – коефіцієнт регуляризації, який визначає баланс між точністю класифікації та складністю моделі.

Такий підхід робить модель більш стійкою до шуму в даних і сприяє узагальненню на нових вибірках, що особливо актуально в контексті виявлення нових, раніше невідомих типів шкідливого ПЗ.

Об’єднавши середню логістичну втрату та регуляризаційний доданок, отримуємо повну функцію ризику, яку необхідно мінімізувати:

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(w,b;x_i',y_i) + \frac{\lambda}{2}\|w\|_2^2, \tag{2.11}$$

Функція (w,b) є строго опуклою, тому її мінімум є єдиним.

Відповідно, задача навчання формулюється як знаходження пари параметрів, що мінімізують цю функцію:

$$(w^*,b^*) = \arg\min_{w,b} J(w,b), \tag{2.12}$$

На практиці формула (2.12) визначає навчальну мету моделі: знайти таке гіперплощинне розділення у просторі ознак, яке з максимальною ймовірністю розділяє класи «malware» і «benign». Метод БФГШ, використаний у коді, реалізує наближену оцінку зворотної матриці Гессе для обчислення напрямку мінімізації, що значно пришвидшує збіжність у порівнянні зі звичайним градієнтним спуском.

Щоб знайти мінімум функції ризику, потрібно обчислити похідні (градієнти) за параметрами w і b. Оскільки функція є диференційовною, її градієнти мають аналітичну форму:

$$\nabla_w J = \frac{1}{n}\sum_{i=1}^{n}(p_\theta(y_i = 1|x_i') - y_i)x_i' + \lambda w, \tag{2.13}$$

$$\frac{\partial b}{\partial J} = \frac{1}{n}\sum_{i=1}^{n}(p_\theta(y_i = 1|x_i') - y_i). \tag{2.14}$$

Перший доданок у (2.13) відповідає похибці прогнозу, а другий – штрафу за великі значення ваг. Ці вирази дозволяють не лише визначати напрямок зменшення втрат, але й будувати квазі-ньютонівське наближення до Гессе, що й використовується в методі БФГШ.

Метод БФГШ є квазі-ньютонівським методом оптимізації, який наближує зворотну матрицю Гессе, уникаючи прямих обчислень другої похідної (що обчислювально дорогі для великих розмірностей). На k-ій ітерації відбувається оновлення параметрів за формулою:

$$\theta_{k+1} = \theta_k - a_k H_k \nabla J(\theta_k), \tag{2.15}$$

де:

$\theta_k = (w_k, b_k)$ – вектор поточних параметрів;

a_k – довжина кроку (визначається методом лінійного пошуку);

H_k – поточне наближення до оберненої матриці Гессе;

$\nabla J(\theta_k)$ – градієнт функції ризику на ітерації k.

Таким чином, метод поступово коригує ваги моделі, рухаючись у напрямку найшвидшого зменшення функції ризику, але з урахуванням локальної кривизни поверхні втрат.

Основна перевага методу БФГШ полягає в ефективному оновленні наближення H_k без обчислення повного Гессе. Оновлення виконується за рекурентним правилом:

$$H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}. \quad (2.16)$$

Ця формула гарантує, що H_{k+1} залишається симетричною й додатно визначеною матрицею, забезпечуючи стабільну збіжність до мінімуму навіть при великих розмірностях простору ознак.

Метод БФГШ продовжує оновлення параметрів доти, доки не виконається одна з умов збіжності:

$$\|\nabla J(\theta_k)\|_2 < \varepsilon \text{ або } |J(\theta_{k+1}) - J(\theta_k)| < \delta\ , \quad (2.17)$$

де, ε та δ – наперед задані малі пороги (наприклад, 10^{-6}).

Перша умова означає, що градієнт майже нульовий (досягнуто стаціонарної точки), а друга – що зміни функції втрат між ітераціями стали незначними, тобто навчання стабілізувалося.

Після завершення навчання отримуємо оптимальні параметри (w^*,b^*), які можна інтерпретувати як внески окремих ознак у ймовірність належності процесу до класу «malware».

Кожен коефіцієнт w_j^* відображає вплив відповідної ознаки x_j' на логарифмічну ймовірність позитивного класу:

$$log\left(\frac{p_\theta(y=1|x')}{1-p_\theta(y=1|x')}\right) = w^{*T}x' + b^*. \quad (2.18)$$

Цей вираз показує, що логарифм відношення шансів (log-odds) є лінійною функцією нормованих ознак.

Знак w_j^* визначає напрям впливу:

якщо $w_j^* > 0$, збільшення x_j^* підвищує ймовірність, що програма є шкідливою;

якщо $w_j^* < 0$, навпаки, зменшує її.

Отже, логістична регресія забезпечує не лише класифікацію, а й аналітичну інтерпретацію важливості кожного параметра.

Регуляризаційний член $\frac{\lambda}{2}\|w\|_2^2$ виконує роль стабілізатора моделі, зменшуючи варіацію параметрів і запобігаючи переобладнанню. Його дія полягає у введенні додаткового штрафу за надто великі ваги.

У термінах мінімізації функції ризику це означає пошук не просто параметрів, які найкраще пояснюють тренувальні дані, а таких, що мінімізують очікувану похибку на нових прикладах:

$$E_{E(x\prime,y)\sim P_{test}}[l(w^*,b^*;x',y)] \approx \frac{1}{n}\sum_{i=1}^{n} l(w^*,b^*;x_i',y_i) + \frac{\lambda}{2}\|w^*\|_2^2. \quad (2.19)$$

Регуляризація таким чином забезпечує узагальнювальну здатність моделі: вона не «запам'ятовує» тренувальні приклади, а формує стійкі закономірності, релевантні і для нових спостережень (зокрема нових зразків ПЗ).

Після навчання модель застосовується до тестових прикладів $\{(x'^{(t)},y^{(t)})\}_{t=1}^{n_{test}}$. Для кожного елемента тестового набору розраховується лінійний скоринг, ймовірність і прогнозований клас:

$$Score^{(t)} = w^{*T}x'^{(t)} + b^*, \quad (2.20)$$

$$p^{(t)} = \sigma(Score^{(t)}) = \frac{1}{1+e^{-Score^{(t)}}}, \quad (2.21)$$

$$\hat{y}^{(t)} = \begin{cases} 1, \text{якщо } p^{(t)} \geq \tau, \\ 0, \text{в іншому випадку} \end{cases}. \quad (2.22)$$

Ця трійка співвідношень формує математичну основу функції виявлення шкідливого ПЗ, тобто класифікація процесів у режимі реального часу.

Щоб узагальнити етап інференсу, можна ввести функцію прийняття рішення $g(\cdot)$, яка відображає простір вхідних даних у простір класів $\{0,1\}$:

$$g(x') = 1\{\sigma(w^{*T}x'^{(t)} + b^*) \geq \tau\}, \quad (2.23)$$

де $1\{\cdot\}$ – індикаторна функція, що повертає 1, якщо умова виконується, і 0 – інакше.

Таким чином, $g(x')$ є детермінованою класифікаційною функцією, яка перетворює безперервну ймовірність на дискретне рішення.

У контексті системи виявлення шкідливого ПЗ ця функція може бути інтегрована в автоматизований процес моніторингу, де результат $g(x')$=1 ініціює подальші дії – ізоляцію, аналіз чи блокування процесу.

Зважаючи на реальні умови експлуатації антивірусних систем, процес виявлення ПЗ має працювати в потоковому режимі, тобто обробляти вхідні дані безперервно в часі. У математичній формі це можна представити як композицію функцій:

$$h_t(z_t) = g\big(T(z_t)\big), t = 1,2, \ldots, \quad (2.24)$$

де:

z_t– поточний сирий запис про процес у момент часу t;

$T(\cdot)$ – оператор підготовки даних;

$g(\cdot)$ – класифікаційна функція.

Таким чином, система перетворюється на безперервний аналітичний оператор $h_t\colon \mathrm{Z} \to \{0,1\}$, який для кожного моменту часу видає рішення – чи є новий процес потенційно шкідливим. Це робить описану модель придатною для інтеграції у реальні середовища захисту інформаційних систем.

2.4 Визначення показників оцінки якості моделі

Оцінювання ефективності побудованої моделі виявлення шкідливого програмного забезпечення здійснюється за допомогою п’яти базових метрик, що відображають різні аспекти її продуктивності: точність класифікації (Accuracy), площа під *ROC*-кривою (*AUC*), гармонійне середнє *Precision* і *Recall* (*F1*- міра), точність позитивного прогнозу (*Precision*) та повнота виявлення (*Recall*).

Ці показники обчислюються на основі результатів тестування моделі після навчання і дозволяють комплексно оцінити якість класифікації як у термінах правильності рішень, так і в термінах їх надійності.

Точність класифікації є базовим показником, який визначає частку правильно класифікованих спостережень серед усіх перевірених прикладів. Вона відображає загальну правильність роботи моделі:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (2.25)$$

де:

TP – кількість правильно виявлених шкідливих зразків (*True Positive*);

TN – кількість коректно розпізнаних безпечних зразків (*True Negative*);

FP – хибні спрацьовування;

FN – пропущені загрози.

Високе значення *Accuracy* означає, що модель у середньому приймає правильне рішення для більшості прикладів, однак ця метрика не враховує співвідношення між класами.

Показник *AUC* (*Area Under the ROC Curve*) є інтегральною характеристикою здатності моделі розрізняти шкідливе та безпечне програмне забезпечення незалежно від порогу прийняття рішення. *ROC*-крива будується у координатах:

$$(FPR, TPR) = \left(\frac{FP}{FP+TN}, \frac{TP}{TP+FN}\right), \quad (2.26)$$

а площа під нею обчислюється як:

$$AUC = \int_0^1 TPR(FPR)d(FPR) \; . \quad (2.27)$$

Чим ближче значення *AUC* до 1, тим вища дискримінативна здатність моделі: вона чітко розділяє класи «malware» та «benign», мінімізуючи хибні класифікації на будь-якому порозі.

Оскільки точність та повнота мають протилежну спрямованість (покращення одного може зменшити інше), для оцінювання їхнього балансу застосовується *F1*-міра – гармонійне середнє значень *Precision* і *Recall*:

$$\mathrm{F1} = 2 \cdot \frac{Precision \cdot Recal}{Precision + Recal} \; . \quad (2.28)$$

Високе значення *F1*-міра свідчить, що модель водночас добре виявляє шкідливе ПЗ і уникає помилкових спрацьовувань. Цей показник є

найінформативнішим для задач із дисбалансом класів, де критично важливо знайти компроміс між виявленням та хибними тривогами.

Метрика *Precision* визначає, яка частка зразків, віднесених моделлю до класу «malware», справді є шкідливою. Вона показує надійність позитивних рішень:

$$\text{Precision} = \frac{TP}{TP+FP}\,. \tag{2.29}$$

Високе значення *Precision* означає, що більшість попереджень моделі обґрунтовані, тобто кількість хибних тривог мінімальна. Цей показник особливо важливий для систем реального часу, де надмірна кількість помилкових блокувань може негативно впливати на стабільність роботи користувача або мережі.

Показник *Recall* (або *True Positive Rate*) оцінює здатність моделі виявляти всі наявні загрози:

$$Recall\ = \frac{TP}{TP+FN}\,. \tag{2.30}$$

Високе значення Recall свідчить, що система майже не пропускає шкідливих об’єктів, що є ключовою вимогою до моделей кіберзахисту.

Зменшення цього показника свідчить про втрату чутливості – частину атак або заражених процесів може бути пропущено.

Комплексне використання описаних метрик дозволяє отримати об’єктивну оцінку якості моделі, виявити можливі компроміси між виявленням і помилковими спрацьовуваннями та визначити оптимальні налаштування порогу прийняття рішення τ для конкретного сценарію застосування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА РЕЗУЛЬТАТІВ

3.1 Вибір мови програмування, середовища розробки та бібліотеки

Побудова програмного забезпечення для аналізу шкідливого програмного забезпечення потребує ретельного вибору інструментів розробки, які забезпечують баланс між продуктивністю, гнучкістю, стабільністю та зручністю реалізації алгоритмів машинного навчання. Вибір мови програмування і середовища визначається типом задачі, обсягом даних, що обробляються, а також можливостями інтеграції з бібліотеками аналітики та візуалізації. Для створення інтелектуальної системи аналізу шкідливого ПЗ особливу увагу приділено мовам Python, Java та C#, оскільки вони широко застосовуються в дослідженнях з кібербезпеки та побудові моделей машинного навчання.

Python – це високорівнева мова програмування з простою синтаксичною структурою, орієнтована на швидку розробку наукових і аналітичних систем [51]. Її основна перевага полягає у великому наборі бібліотек для машинного навчання, зокрема Scikit-learn, TensorFlow та PyTorch, які забезпечують повний цикл від підготовки даних до побудови моделей. Завдяки відкритому екосередовищу Python активно використовується у прототипуванні систем виявлення загроз і для швидкої валідації алгоритмів.

Java – універсальна об’єктно-орієнтована мова, що характеризується високою стабільністю, кросплатформеністю та розвинутими засобами багатопотокового програмування [52]. Вона застосовується у великих корпоративних системах безпеки, де важливими є продуктивність і сумісність із різними платформами. Бібліотеки, такі як Weka або Deeplearning4j, забезпечують реалізацію алгоритмів аналізу даних у середовищі Java для задач класифікації та прогнозування.

C# – мова програмування від Microsoft, що поєднує потужність об’єктно-орієнтованого підходу з широкими можливостями розробки настільних і

серверних додатків [53]. Завдяки інтеграції з бібліотекою ML.NET вона забезпечує реалізацію алгоритмів машинного навчання безпосередньо в екосистемі .NET, що спрощує розробку безпечних аналітичних систем. C# відзначається високою продуктивністю, зручною інтеграцією з базами даних та зручним графічним інтерфейсом, що робить його придатним для побудови прикладних систем аналізу шкідливого ПЗ.

У табл. 3.1 наведено порівняльну характеристику мов розглянутих програмування за основними якісними та кількісними показниками, що визначають їх ефективність у розробленні аналітичних систем для аналізу шкідливого програмного забезпечення.

Таблиця 3.1.

Порівняльна характеристика мов програмування

№	Показник	Python	Java	C#
1	Тип парадигми	Інтерпретована, динамічна	Компіляційна, об’єктно-орієнтована	Компіляційна, об’єктно-орієнтована
2	Продуктивність виконання (умовна оцінка, MIPS)	~60–80	~120–150	~160–180
3	Рівень оптимізації пам’яті	Середній (динамічна типізація)	Високий	Високий, завдяки CLR
4	Безпечність виконання (система типів, винятки)	Середня	Висока	Висока
5	Зручність інтеграції з базами даних	Висока (ORM-бібліотеки)	Висока (JDBC, Hibernate)	Дуже висока (ADO.NET)
6	Середня кількість рядків коду для реалізації базового алгоритму	35–40	45–50	30–35
7	Підтримка паралельних обчислень	Обмежена (через GIL)	Розвинена (Threads, Executors)	Розвинена (Tasks, async/await)

8	Легкість налагодження та тестування	Висока	Середня	Висока

Вибір мови програмування C# для розробки системи аналізу шкідливого програмного забезпечення обумовлений її здатністю забезпечувати високу продуктивність виконання, стабільність та надійність програмного коду. Завдяки механізму Common Language Runtime (CLR) C# ефективно керує пам'яттю, мінімізуючи ризик витоків і забезпечуючи безпечне виконання операцій із даними. Розвинуті засоби інтеграції з базами даних через ADO.NET спрощують оброблення великих обсягів інформації та підвищують ефективність взаємодії між компонентами системи. Мова підтримує сучасні механізми паралельних обчислень, що є важливим для реалізації аналітичних процесів у реальному часі. Зручність налагодження, висока читабельність коду та інтеграція з середовищем Visual Studio роблять C# оптимальним вибором для побудови комплексної, надійної та продуктивної системи виявлення шкідливого ПЗ.

Після визначення мови програмування C# наступним кроком є вибір оптимального середовища розробки, яке забезпечує можливість роботи з бібліотеками машинного навчання та зручну інтеграцію з базами даних. Для реалізації системи аналізу шкідливого програмного забезпечення розглядалися три найпоширеніші середовища: Visual Studio 2022, Visual Studio Code та JetBrains Rider, які охоплюють широкий спектр потреб – від повноцінної розробки великих проєктів до легких сценаріїв тестування і налагодження. Порівняння цих середовищ дозволяє визначити, яке з них найбільш придатне для розроблення аналітичної системи в рамках обраної технологічної платформи .NET.

Visual Studio 2022 – це повнофункціональне інтегроване середовище розробки від Microsoft, оптимізоване для створення застосунків на C#, .NET, ASP.NET та інших технологіях екосистеми Windows [54]. Воно підтримує потужний налагоджувач, інтелектуальну підсвітку коду та засоби профілювання продуктивності, що особливо важливо для оптимізації аналітичних обчислень. Завдяки інтеграції з ML.NET та SQL Server Visual Studio 2022 забезпечує повний

цикл створення системи – від реалізації алгоритмів машинного навчання до розгортання готового програмного продукту.

Visual Studio Code – це легке, кросплатформне середовище, яке підтримує широкий спектр розширень і дозволяє розробникам працювати з різними мовами програмування, включно з C# [55]. Його гнучкість полягає у можливості налаштування середовища під конкретні потреби за допомогою плагінів, таких як C# Dev Kit і .NET Runtime Tools. Попри відсутність повної інтеграції з інструментами .NET, VS Code є зручним вибором для швидкого тестування, редагування коду та розробки компактних компонентів системи.

JetBrains Rider – це сучасне середовище розробки, створене на базі платформи IntelliJ, яке підтримує технології .NET, ASP.NET, Entity Framework і Xamarin [56]. Воно забезпечує високу швидкодію, зручну систему рефакторингу коду та інтегровані засоби роботи з базами даних. Завдяки глибокій інтеграції з системами контролю версій і кросплатформеній архітектурі, Rider є ефективним вибором для командної розробки масштабних проєктів у сфері кібербезпеки та аналізу даних.

У табл. 3.2 наведено порівняльну характеристику середовищ Visual Studio 2022, Visual Studio Code та JetBrains Rider за ключовими показниками, що відображають їх технічні й експлуатаційні властивості.

Таблиця 3.2.

Порівняльна характеристика середовищ розробки

№	Показник	Visual Studio 2022	Visual Studio Code	JetBrains Rider
1	2	3	4	5
1	Тип середовища	Повнофункціональне IDE	Легке текстове середовище з розширеннями	Інтелектуальне IDE на базі IntelliJ
2	Підтримка мов програмування	C#, C++, Python, F#, JavaScript тощо	30+ мов через плагіни	C#, F#, VB.NET, JavaScript

3	Підтримка баз даних	Інтеграція з SQL Server, SQLite, MySQL	Обмежена (через розширення)	Вбудовані інструменти роботи з БД
4	Підтримка профілювання коду	Вбудована (Performance Profiler, Diagnostic Tools)	Відсутня	Є, але менш гнучка

Продовження таблиці 3.2

1	2	3	4	5
5	Продуктивність при великих проєктах	Висока, завдяки оптимізації під .NET і багатопотоковість	Середня	Висока, але потребує більше пам'яті
6	Інтеграція з ML-бібліотеками	Повна підтримка ML.NET, ONNX, CNTK	Обмежена (TensorFlow, через розширення)	Часткова (через зовнішні плагіни)
7	Рівень інтеграції з системою контролю версій	Повна (вбудована)	Часткова (через плагіни)	Повна
8	Підтримка профілювання коду	Вбудована (Performance Profiler, Diagnostic Tools)	Відсутня	Є, але менш гнучка
9	Можливості розгортання застосунків	Повна (Desktop, Web, Cloud, Container)	Обмежена	Повна
10	Середня швидкість компіляції (для проєктів на C#, с)	8–10	12–14	10–12

Вибір середовища Visual Studio 2022 для реалізації системи аналізу шкідливого програмного забезпечення зумовлений його функціональною повнотою, високим рівнем інтеграції з екосистемою .NET і підтримкою сучасних інструментів машинного навчання. Це середовище забезпечує стабільну роботу з великими проєктами завдяки багатопотоковій обробці та оптимізації компілятора, що скорочує час збірки й підвищує ефективність тестування. Вбудовані засоби профілювання коду, моніторингу пам'яті та покрокового налагодження дозволяють точно локалізувати потенційні помилки та оптимізувати обчислювальні процеси. Окрім того, Visual Studio 2022 має гнучку

інтеграцію з SQL Server, ADO.NET і ML.NET, що дає змогу реалізовувати аналітичні модулі без необхідності використання сторонніх інструментів. Завдяки широким можливостям розгортання на платформах Desktop, Web і Cloud, це середовище забезпечує повний цикл розроблення, тестування та впровадження систем безпеки на базі штучного інтелекту.

Після визначення мови програмування та середовища розробки важливим етапом є вибір бібліотек машинного навчання, які забезпечують реалізацію алгоритмів класифікації, оброблення даних і побудову моделей для аналізу шкідливого програмного забезпечення. Вибір бібліотеки визначається критеріями продуктивності, зручності інтеграції з платформою .NET, доступністю методів попередньої обробки даних та підтримкою різних типів моделей. Для розроблення аналітичної системи були розглянуті три найпоширеніші бібліотеки: ML.NET, Accord.NET та Encog, які поєднують стабільність, відкритість і широкі можливості для побудови інтелектуальних моделей.

ML.NET – це фреймворк машинного навчання, розроблений компанією Microsoft, який дозволяє створювати, навчати та впроваджувати моделі безпосередньо у середовищі .NET без необхідності використання сторонніх мов [57]. Він підтримує алгоритми класифікації, регресії, кластеризації та детекції аномалій, а також забезпечує сумісність з форматами ONNX і TensorFlow. Завдяки інтеграції з Visual Studio та SQL Server, ML.NET дозволяє легко реалізувати повний аналітичний цикл – від підготовки даних до прогнозування в режимі реального часу.

Accord.NET – це універсальний фреймворк для наукових обчислень і обробки сигналів у середовищі .NET, який містить широкий набір алгоритмів машинного навчання, статистики та комп’ютерного зору [58]. Його особливістю є поєднання математичних інструментів із засобами для роботи з аудіо- та візуальними даними, що робить бібліотеку зручною для дослідницьких і аналітичних застосунків. Accord.NET активно використовується для створення систем класифікації та виявлення патернів у великих масивах даних.

Encog – це бібліотека для побудови нейронних мереж і реалізації еволюційних алгоритмів, яка підтримує навчання як у режимі CPU, так і GPU [59]. Вона надає інструменти для створення багатошарових перцептронів, рекурентних і згорткових мереж, що робить її придатною для моделювання складних залежностей у даних. Encog характеризується високою швидкістю навчання та компактністю коду, завдяки чому ефективно використовується у задачах прогнозування та розпізнавання шкідливого програмного забезпечення.

У табл. 3.3 наведено узагальнену порівняльну характеристику бібліотек, що використовуються у середовищі C# для реалізації моделей машинного навчання.

Таблиця 3.3.

Порівняльна характеристика бібліотек машинного навчання у середовищі C#

№	Показник	ML.NET	Accord.NET	Encog
1	Розробник / підтримка	Microsoft	Незалежна спільнота	Heaton Research
2	Підтримка середовища .NET	Повна (.NET 6/7/8)	Повна (.NET Framework, .NET Core)	Часткова (.NET Framework, Mono)
3	Формати моделей	ONNX, TensorFlow, ZIP	Власний двійковий формат	Власний текстовий формат
4	Інтеграція з Visual Studio	Повна (вбудований ML.NET Model Builder)	Часткова (через бібліотеки NuGet)	Відсутня
5	Продуктивність навчання (умовна оцінка)	Висока	Середня	Висока при GPU
6	Можливість розгортання в реальному часі	Так	Обмежено	Так
7	Простота у використанні	Висока	Середня	Середня

8	Підтримка GPU	Часткова (через ONNX/TensorFlow integration)	Відсутня	Повна (CUDA/OpenCL)
9	Придатність до реалізації систем кібербезпеки	Висока	Висока	Висока при роботі з нейромережами

Вибір бібліотеки ML.NET для розробки системи аналізу шкідливого програмного забезпечення обумовлений її повною інтеграцією з екосистемою .NET та середовищем Visual Studio, що забезпечує зручність побудови, навчання і тестування моделей без залучення зовнішніх інструментів. Бібліотека підтримує сучасні формати моделей, зокрема ONNX і TensorFlow, що дозволяє легко поєднувати власні алгоритми з уже наявними рішеннями машинного навчання. Завдяки високій продуктивності, стабільності та можливості розгортання моделей у режимі реального часу ML.NET є придатною для створення аналітичних систем, здатних швидко реагувати на нові типи загроз. Простота у використанні та офіційна підтримка Microsoft гарантують надійність, актуальність і довгострокову підтримку побудованого програмного продукту.

3.2 Побудова основних алгоритмів системи

Під час розроблення програмного забезпечення для аналізу шкідливого програмного забезпечення ключовим етапом стала побудова алгоритмів, що забезпечують послідовне виконання процесів навчання, оцінювання та збереження моделі. Формування цих алгоритмів здійснювалося з урахуванням принципів модульності, перевірки коректності вхідних даних і надійності оброблення результатів. Особлива увага приділялася автоматизації взаємодії між компонентами системи, що дозволяє мінімізувати вплив людського фактора та забезпечити стабільність процесу навчання моделей у різних умовах експлуатації.

На рис. 3.1 наведено блок-схему алгоритму навчання моделі, яка відображає логіку послідовних дій під час побудови інтелектуального модуля

класифікації. Процес розпочинається з вибору навчального набору, після чого система перевіряє коректність вхідного файлу. У разі виявлення помилки користувачу виводиться відповідне повідомлення, тоді як при наявності валідного файлу відбувається завантаження даних до середовища оброблення. Далі формується конвеєр обробки, що включає нормалізацію ознак та підготовку даних до навчання. Після цього запускається процес побудови моделі, який завершується її оцінюванням за відповідними метриками якості.

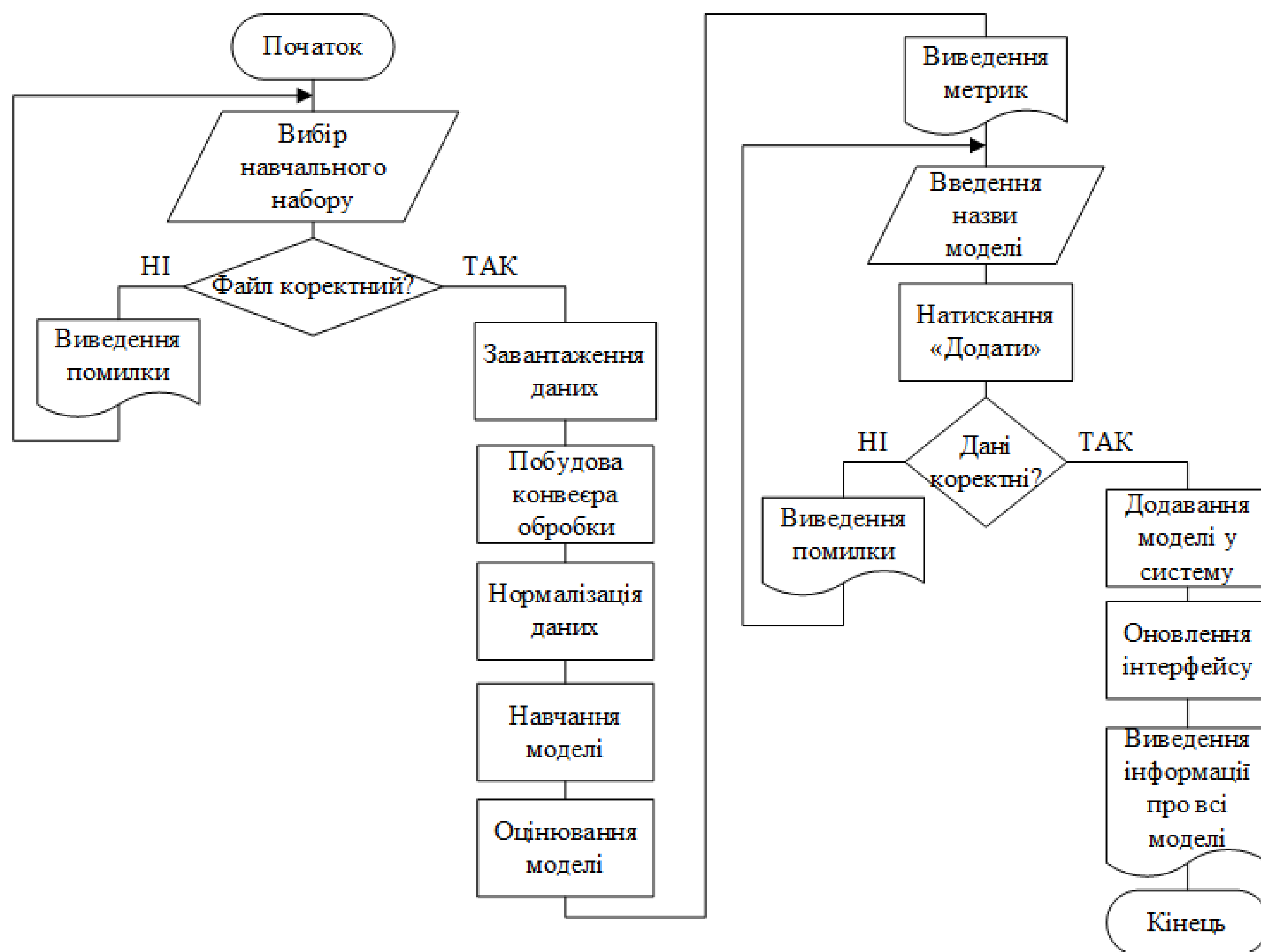

Рисунок 3.1 – Блок-схема алгоритму навчання моделі

Наступним етапом є взаємодія користувача із системою, де після виведення метрик користувач вводить назву моделі та підтверджує дію натисканням кнопки «Додати». Якщо введені дані виявляються некоректними, система повідомляє про помилку. У протилежному випадку модель додається до бази даних, інтерфейс оновлюється, і користувач отримує оновлену інформацію про всі доступні моделі. Таким чином, поданий алгоритм забезпечує повний

замкнений цикл – від завантаження даних і навчання до інтеграції моделі в систему та її подальшого використання для прогнозування загроз.

На рисунку 3.2 подано блок-схему алгоритму моніторингу та виявлення шкідливого програмного забезпечення, що реалізує логіку роботи системи під час її експлуатації. Алгоритм розпочинається з етапу ініціалізації змінних і створення необхідних об’єктів, які забезпечують доступ до бази даних, моделей машинного навчання та інтерфейсних елементів. Після цього виконується завантаження списку доступних моделей, що зберігаються в системі, і користувачу надається можливість вибору потрібної моделі для подальшої роботи. Система перевіряє наявність обраної моделі, і якщо вона відсутня або пошкоджена, відбувається виведення повідомлення про помилку, що запобігає некоректному виконанню наступних етапів.

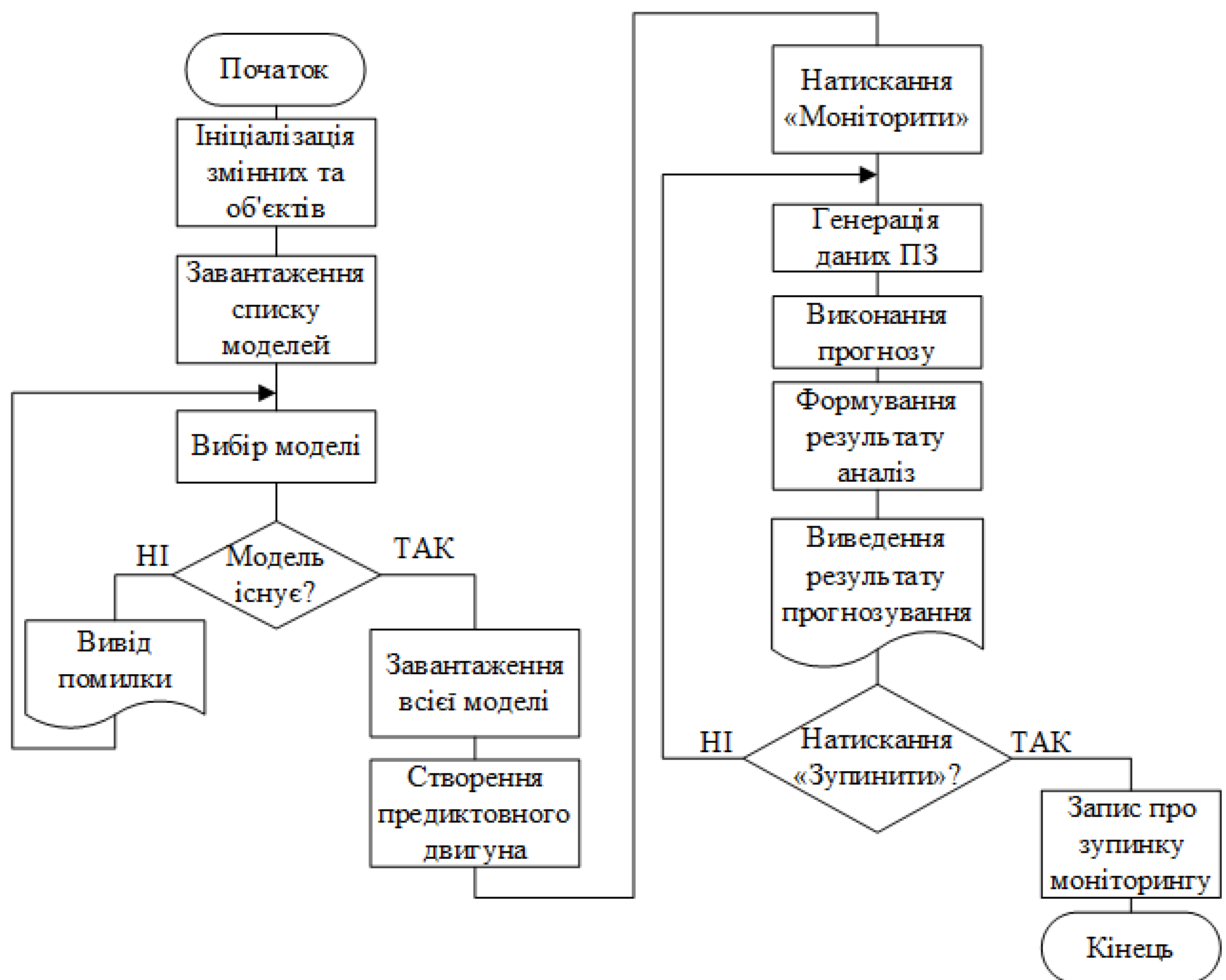

Рисунок 3.2 – Блок-схема моніторингу та виявлення шкідливого ПЗ

У випадку, коли модель існує і є доступною, відбувається її повне завантаження в оперативну пам’ять та створення предиктивного двигуна, який забезпечує обчислення прогнозів у режимі реального часу. Після цього користувач активує процес моніторингу натисканням кнопки «Моніторити», що ініціює генерацію вхідних даних про програмні процеси, їх попередню обробку та передачу у модель машинного навчання. На основі отриманих ознак система здійснює прогнозування та формує структурований результат аналізу, який містить висновки щодо наявності або відсутності ознак шкідливого програмного забезпечення.

Після оброблення результатів користувачу виводиться підсумкова інформація у зручному текстовому форматі, яка може містити ймовірність виявлення загрози, ідентифікатор файлу, рівень ризику тощо. Моніторинг триває доти, доки користувач не ініціює його завершення через натискання кнопки «Зупинити». У цей момент система фіксує подію завершення роботи, оновлює журнали спостережень і переходить у стан очікування нової сесії. Така організація алгоритму забезпечує безперервне спостереження за поведінкою програмного забезпечення з можливістю оперативного реагування на загрози, що виникають у реальному часі.

3.3 Реалізація ПЗ для навчання, збереження та використання моделі

Визначене технологічне середовище та обрана бібліотека машинного навчання стали основою для переходу до етапу реалізації програмного забезпечення, у межах якого було створено комплекс функціональних модулів, що забезпечують повний цикл роботи з моделлю – від її навчання до збереження та подальшого використання у процесі аналізу шкідливого програмного забезпечення. У процесі розробки передбачено створення окремих класів для роботи з даними, які відповідають за ініціалізацію, оброблення, збереження та отримання інформації про моделі з бази даних. Вони формують єдиний інтерфейс доступу між бізнес-логікою та рівнем даних, що сприяє впорядкованій структурі коду та спрощує подальшу підтримку і розширення системи. Завдяки

чітко визначеним методам і параметрам взаємодії між компонентами забезпечується узгодженість роботи програмних модулів і стабільність функціонування всієї системи в умовах реального навантаження.

На рис. 3.3 наведено фрагмент методу, що реалізує збереження параметрів навченої моделі у базі даних. Даний метод відповідає за створення нового запису у таблиці Models, у якому фіксуються назва моделі, дата її створення та посилання на файл, що містить саму модель у серіалізованому вигляді.

```
public void InsertModels(string ModelsName, string ModelsFileModel) {
    SqlConnection connection = new SqlConnection(_ConnString);
    string query = "INSERT into Models (ModelsName, CreateDate, ModelsFileModel) " +
      "VALUES (@ModelsName, @CreateDate, @ModelsFileModel)";
    SqlCommand command = new SqlCommand(query, connection);
    command.Parameters.AddWithValue("@ModelsName", ModelsName);
    command.Parameters.AddWithValue("@CreateDate", DateTime.Now);
    command.Parameters.AddWithValue("@ModelsFileModel", ModelsFileModel);
    connection.Open();
    command.ExecuteNonQuery();
    connection.Close();
}
```

Рисунок 3.3 – Метод збереження даних моделі

Метод збереження даних моделі побудований на основі засобів роботи з SQL Server, що забезпечує високу стабільність і контроль над транзакціями. Для встановлення з'єднання використовується об'єкт SqlConnection, який ініціалізується за допомогою заздалегідь визначеного рядка підключення. Далі формується SQL-запит на вставку запису до таблиці, у якому параметри моделі передаються через об'єкт SqlCommand із використанням параметризованих запитів. Це дозволяє уникнути ризику SQL-ін'єкцій і гарантує безпечне внесення даних до бази. Після відкриття з'єднання виконується команда вставки, а по завершенні операції – з'єднання закривається.

Під час розробки інтерфейсу користувача було реалізовано інтуїтивно зрозумілів форми взаємодії з системою, яка дозволяє виконувати основні операції безпосередньо через графічні елементи керування. Головна форма містить елементи для завантаження даних, запуску навчання, збереження моделей і відображення результатів, що забезпечує зручність у роботі навіть для

користувачів без глибоких технічних знань. Особливу увагу приділено інтерактивності інтерфейсу – усі операції виконуються асинхронно, щоб уникнути блокування програми під час виконання тривалих процесів, таких як завантаження файлів або оброблення великих наборів даних.

Метод, що наведений на рис. 3.4, реалізує асинхронний обробник події натискання кнопки «Відкрити». Його основним завданням є забезпечення вибору файлу, який містить вхідні дані для подальшого аналізу або навчання моделі.

```
private async void OpenBtn_Click(object sender, EventArgs e) {
  // 1) Вибір файлу
  using (var openFileDialog = new OpenFileDialog {
    Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*",
    FilterIndex = 1,
    RestoreDirectory = true,
    Title = "Оберіть CSV з даними"
  }) {
    if (openFileDialog.ShowDialog() != DialogResult.OK)
      return;

    _Path = openFileDialog.FileName;
    FileNameTBox.Text = openFileDialog.FileName;
  }
```

Рисунок 3.4 – Аинхронний обробник події натискання кнопки «Відкрити»

При виклику функції відкривається стандартне діалогове вікно вибору файлів, у якому користувач може обрати CSV-документ або інший формат, допустимий системою. Встановлені параметри діалогу визначають фільтр дозволених розширень, збереження попереднього каталогу та заголовок вікна, що підвищує зручність роботи. Після підтвердження вибору шлях до файлу зберігається у змінній, а його назва відображається у відповідному полі інтерфейсу, забезпечуючи користувача візуальним підтвердженням успішного завантаження даних. Завдяки асинхронній структурі обробки подій інтерфейс залишається активним, що гарантує стабільність та комфортність взаємодії під час виконання файлових операцій.

У наведеному фрагменті коду на рис. 3.5 реалізовано процес ініціалізації контексту машинного навчання, який є базовим елементом будь-якої програми, що використовує бібліотеку ML.NET.

```
try {
  // 2) Ініціалізація контексту
  mlContext = new MLContext(seed: 123);
  AppendRaport("Ініціалізовано ML-контекст.\r\n");
```

Рисунок 3.5 – Процес ініціалізації контексту машинного навчання

Створення об’єкта MLContext забезпечує формування середовища, у якому виконуються всі основні операції – завантаження даних, побудова конвеєрів оброблення, навчання моделі та оцінювання результатів. Під час ініціалізації задається фіксоване значення seed, що гарантує відтворюваність експериментів та стабільність результатів при повторному виконанні навчання.

На рис. 3.6 наведенон процес реалізації асинхронного завантаження вхідних даних із вибраного файлу для подальшої обробки в середовищі ML.NET.

```
// 3) Завантаження даних
var rawData = await Task.Run(() =>
    mlContext.Data.LoadFromTextFile<MalwareRow>(
        path: _Path, hasHeader: true, separatorChar: ','));
AppendRaport("Дані завантажено. Формую булеву мітку Label…\r\n");
```

Рисунок 3.6 – Процес ініціалізації контексту машинного навчання

За допомогою методу LoadFromTextFile дані з CSV-файлу імпортуються у формат, сумісний із типом MalwareRow, який визначає структуру одного рядка набору даних, тобто набір ознак, що описують властивості програмного об’єкта. Використання асинхронного виклику через Task.Run дозволяє виконувати завантаження у фоновому режимі, не блокуючи головний інтерфейс програми, що є особливо важливим при роботі з великими файлами.

На рис. 3.7 наведено фрагмент коду у якому здійснюється формування набору даних із логічною міткою, необхідною для виконання класифікації шкідливого програмного забезпечення.

```
var labeledEnum = mlContext.Data.CreateEnumerable<MalwareRow>(rawData, reuseRowObject: false)
    .Select(r => new MalwareRowLabeled {
      hash = r.hash,
      millisecond = r.millisecond,
      classification = r.classification,
      state = r.state, usage_counter = r.usage_counter, prio = r.prio,
      static_prio = r.static_prio, normal_prio = r.normal_prio, policy = r.policy,
      vm_pgoff = r.vm_pgoff, vm_truncate_count = r.vm_truncate_count,
      task_size = r.task_size, cached_hole_size = r.cached_hole_size,
      free_area_cache = r.free_area_cache, mm_users = r.mm_users,
      map_count = r.map_count, hiwater_rss = r.hiwater_rss, total_vm = r.total_vm,
      shared_vm = r.shared_vm, exec_vm = r.exec_vm, reserved_vm = r.reserved_vm,
      nr_ptes = r.nr_ptes, end_data = r.end_data, last_interval = r.last_interval,
      nvcsw = r.nvcsw, nivcsw = r.nivcsw, min_flt = r.min_flt, maj_flt = r.maj_flt,
      fs_excl_counter = r.fs_excl_counter, @lock = r.@lock, utime = r.utime,
      stime = r.stime, gtime = r.gtime, cgtime = r.cgtime, signal_nvcsw = r.signal_nvcsw,
      Label = ((r.classification ?? string.Empty).Trim().ToLowerInvariant() == "malware")
    });
```

Рисунок 3.7 – Формування набору даних із логічною міткою

На основі раніше завантажених сирих даних створюється нова колекція об’єктів типу MalwareRowLabeled, кожен з яких містить повний набір ознак, що характеризують поведінку або властивості програмного процесу. За допомогою операції Select для кожного запису формується додаткове поле Label, яке набуває значення true у випадку, якщо атрибут classification містить ознаку «malware», та false – якщо об’єкт належить до безпечного програмного забезпечення. Таким чином, здійснюється підготовка навчальної вибірки з бінарною міткою, яка використовується моделлю для розрізнення шкідливих і легітимних програм. Такий підхід забезпечує однозначне визначення цільової змінної й формує узгоджену структуру даних для подальшого етапу навчання алгоритму машинного навчання.

У фрагменті програмного коду на рис. 3.8 реалізовано етап розділення початкового набору даних на дві частини – навчальну та тестову вибірки, що є обов’язковою процедурою у процесі побудови та перевірки моделей машинного навчання.

```
// 5) Поділ на train/test
var split = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2, seed: 123);
AppendRaport("Виконано поділ Train/Test (80/20).\r\n");
```

Рисунок 3.8 – Розділення початкового набору даних

Використовується метод TrainTestSplit, який автоматично формує два піднабори даних: 80 % для навчання моделі та 20 % для оцінювання її якості на незалежних прикладах. Застосування фіксованого параметра seed забезпечує відтворюваність результатів при повторних запусках експериментів, що важливо для контролю стабільності алгоритму. Після виконання операції система додає відповідне повідомлення до звіту, інформуючи користувача про успішне завершення поділу вибірки та готовність даних до етапу навчання.

На рис. 3.9 наведено формування конвеєра машинного навчання, який об’єднує всі етапи оброблення даних у єдину логічну послідовність. На першому етапі визначається набір вхідних ознак – featureColumns, що містить параметри, які характеризують поведінку програмних процесів: час виконання, системні лічильники, параметри пам’яті, пріоритети потоків тощо. Ці дані формують багатовимірний простір ознак, у якому модель аналізує взаємозв’язки між характеристиками шкідливого та легітимного програмного забезпечення. Далі відбувається побудова самого конвеєра, який послідовно виконує видалення нерелевантних атрибутів, об’єднання вибраних ознак у єдиний вектор Features, нормалізацію значень методом MinMax та підключення класифікатора логістичної регресії для навчання моделі.

```
var featureColumns = new[]
{
    nameof(MalwareRow.millisecond), nameof(MalwareRow.state), nameof(MalwareRow.usage_counter),
    nameof(MalwareRow.prio), nameof(MalwareRow.static_prio), nameof(MalwareRow.normal_prio),
    nameof(MalwareRow.policy), nameof(MalwareRow.vm_pgoff), nameof(MalwareRow.vm_truncate_count),
    nameof(MalwareRow.task_size), nameof(MalwareRow.cached_hole_size), nameof(MalwareRow.free_area_cache),
    nameof(MalwareRow.mm_users), nameof(MalwareRow.map_count), nameof(MalwareRow.hiwater_rss),
    nameof(MalwareRow.total_vm), nameof(MalwareRow.shared_vm), nameof(MalwareRow.exec_vm),
    nameof(MalwareRow.reserved_vm), nameof(MalwareRow.nr_ptes), nameof(MalwareRow.end_data),
    nameof(MalwareRow.last_interval), nameof(MalwareRow.nvcsw), nameof(MalwareRow.nivcsw),
    nameof(MalwareRow.min_flt), nameof(MalwareRow.maj_flt), nameof(MalwareRow.fs_excl_counter),
    nameof(MalwareRow.@lock), nameof(MalwareRow.utime), nameof(MalwareRow.stime),
    nameof(MalwareRow.gtime), nameof(MalwareRow.cgtime), nameof(MalwareRow.signal_nvcsw)
};
AppendRaport("Побудова конвеєра…\r\n");
var pipeline = mlContext.Transforms.DropColumns(nameof(MalwareRowLabeled.hash),
                                                nameof(MalwareRowLabeled.classification))
    .Append(mlContext.Transforms.Concatenate("Features", featureColumns))
    .Append(mlContext.Transforms.NormalizeMinMax("Features"))
    .Append(mlContext.BinaryClassification.Trainers.LbfgsLogisticRegression(
        labelColumnName: "Label", featureColumnName: "Features"));
```

Рисунок 3.9 – Формування конвеєра машинного навчання

Після побудови конвеєра ініціюється етап навчання моделі на основі підготовленої навчальної вибірки. Використовується асинхронний виклик, який дозволяє запускати процес тренування у фоновому режимі без блокування основного інтерфейсу користувача. Алгоритм логістичної регресії оптимізує вагові коефіцієнти для кожної з ознак, формуючи математичну залежність між параметрами поведінки програми та її належністю до певного класу – шкідливого або безпечного. Після завершення навчання модель зберігає отриману структуру, готову до використання для подальших прогнозів та тестування на нових даних.

На рис. 3.10 наведено процес оцінювання навченої моделі, що дає змогу визначити її якість і здатність до правильного розпізнавання шкідливого програмного забезпечення. Спочатку модель застосовується до тестової вибірки, утворюючи прогнозовані значення за допомогою методу Transform, після чого отримані результати проходять аналітичну перевірку через функцію Evaluate. Ця функція обчислює ключові метрики для бінарної класифікації, такі як точність, площа під кривою ROC, F1-показник, точність позитивних передбачень та повноту, що дозволяє всебічно оцінити здатність моделі відрізняти шкідливі об’єкти від безпечних.

```
// 8) Оцінка
AppendRaport("Оцінювання моделі…\r\n");
var predictions = trainedModel.Transform(split.TestSet);
var metrics = mlContext.BinaryClassification.Evaluate(
    data: predictions, labelColumnName: "Label", scoreColumnName: "Score");

// 9) Вивід результатів
AppendRaport("\r\n=== РЕЗУЛЬТАТИ ===\r\n");
AppendRaport($"Accuracy  : {metrics.Accuracy:P2}\r\n");
AppendRaport($"AUC (ROC) : {metrics.AreaUnderRocCurve:P2}\r\n");
AppendRaport($"F1-Score  : {metrics.F1Score:P2}\r\n");
AppendRaport($"Precision : {metrics.PositivePrecision:P2}\r\n");
AppendRaport($"Recall    : {metrics.PositiveRecall:P2}\r\n");
```

Рисунок 3.10 – Процес оцінювання навченої моделі

Після завершення оцінювання результати виводяться до звіту, формуючи підсумкову аналітичну частину експерименту. Значення метрик подаються у

зрозумілому форматі з точністю до сотих, що полегшує їх інтерпретацію під час аналізу ефективності алгоритму.

У поданому на рис. 3.11 методі реалізовано логіку збереження навченої моделі після завершення процесу навчання та її успішної перевірки.

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    //Зберігання моделі
    string pathName = @"\teach\" + GenerateFileName() + ".zip";
    string localProj =
      System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
    _ModelsProvider.InsertModels(ModelsNamesTBox.Text, pathName);
    mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
    ClearAllData();
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було навчено модель " +
      ModelsNamesTBox.Text, DateTime.Now);
    MessageBox.Show("Дані успішно збережено!");
  }
}
```

Рисунок 3.11 – Метод збереження навченої моделі

На початку виконується перевірка коректності введених користувачем даних, після чого формується унікальна назва файлу моделі, який буде збережено у підкаталозі teach у межах робочої директорії проєкту. Метод InsertModels забезпечує внесення інформації про створену модель до бази даних – зокрема, її назви та шляху до файлу, що дозволяє відстежувати історію навчань і використовувати моделі повторно. За допомогою засобів бібліотеки ML.NET модель серіалізується у файл формату .zip, що містить як параметри навченої структури, так і метадані про схему вхідних даних.

За допомогою засобів бібліотеки ML.NET модель серіалізується у файл формату .zip, що містить як параметри навченої структури, так і метадані про схему вхідних даних. Після завершення збереження виконується очищення полів інтерфейсу для підготовки до нових операцій та автоматичне занесення відповідного запису у журнал дій користувачів через метод InsertLogs, що фіксує факт створення моделі. Завершальним етапом є виведення повідомлення про успішне збереження, яке інформує користувача про коректне завершення операції та гарантує готовність системи до подальшого використання моделі.

У наведеному на рис. 3.12 методі реалізується завантаження раніше навченої моделі машинного навчання для подальшого використання у процесі прогнозування.

```
private void LoadData(string FilePath) {
  string localProj = Application.StartupPath + FilePath;
  // Визначте DataViewSchema для конвеєра підготовки даних і навченої моделі
  DataViewSchema modelSchema;
  // Завантаження моделі
  ITransformer model = context.Model.Load(localProj, out modelSchema);
  //Створення механізму прогнозування
  predictionEngine =
    context.Model.CreatePredictionEngine<MalwareRow,
    MalwarePrediction>(model);
}
```

Рисунок 3.12 – Завантаження раніше навченої моделі машинного навчання

Формується абсолютний шлях до файлу моделі на основі робочого каталогу програми та переданого параметра FilePath, що дозволяє забезпечити коректне звернення до збереженого ресурсу незалежно від місця запуску застосунку. Далі визначається об’єкт DataViewSchema, який містить метадані про структуру даних, з якою працювала модель під час навчання.

Після цього за допомогою методу Load відбувається десеріалізація моделі з файлу, у результаті чого створюється об’єкт типу ITransformer, який представляє повністю готовий до роботи конвеєр оброблення даних. На основі завантаженої моделі формується об’єкт PredictionEngine, що забезпечує можливість здійснювати одиничні передбачення в режимі реального часу.

Рис. 3.13 відображає фрагмент програмного коду, у якому реалізовано процес прогнозування результатів класифікації шкідливого програмного забезпечення на основі навченої моделі. За допомогою методу Predict об’єкт predictionEngine виконує обчислення для переданого зразка sample, що містить набір характеристик програмного процесу. Отримане передбачення містить не лише класову мітку, а й числову оцінку ймовірності належності об’єкта до шкідливого ПЗ. Обчислене значення ймовірності переводиться у відсоткову форму, а функція ClampFloat гарантує, що воно залишатиметься в межах від 0 до 100, запобігаючи відхиленням, спричиненим похибками обчислень.

```
    var prediction = predictionEngine.Predict(sample);
    float prob = ClampFloat(prediction.Probability * 100f, 0f, 100f);
    var sb = new StringBuilder();
    sb.AppendLine("\r\n--- РЕЗУЛЬТАТ ПРОГНОЗУ ---");
    if (!string.IsNullOrWhiteSpace(sample.hash))
      sb.AppendLine($"Hash: {sample.hash}");
    if (!string.IsNullOrWhiteSpace(sample.classification))
      sb.AppendLine($"Оригінальна мітка (CSV): {sample.classification}");
    if (prediction.PredictedLabel)
      sb.AppendLine($"Виявлено шкідливе ПЗ (malware)\r\nЙмовірність: {prob:0.00}%");
    else
      sb.AppendLine($"Безпечне ПЗ (benign)\r\nЙмовірність: {(100f - prob):0.00}%");

    sb.AppendLine($"Score: {prediction.Score:0.0000}");
    MonitoringTBox.Text += sb.ToString();
}
```

Рисунок 3.13 – Процес прогнозування результатів класифікації шкідливого ПЗ

Формується структурований текстовий звіт із результатами прогнозу, який виводиться у вікні моніторингу системи. До звіту додаються ідентифікаційні дані зразка, зокрема hash та початкова класифікація з CSV-файлу, після чого відображається висновок моделі про належність програми до шкідливих або безпечних. Якщо прогноз позитивний, система повідомляє про виявлення шкідливого ПЗ та виводить рівень довіри у відсотках, тоді як при відсутності загрози показується зворотне значення ймовірності. Додатково зазначається числовий показник Score, що відображає ступінь упевненості моделі, а сформований звіт додається до текстового поля MonitoringTBox, забезпечуючи користувачу повну картину результатів аналізу.

3.4 Проведення експериментів на основі реальних зразків

Для оцінювання працездатності та перевірки точності розробленої системи класифікації шкідливого програмного забезпечення було проведено серію експериментів на основі реальних зразків даних. Як навчальний набір для побудови моделі машинного навчання використано відкритий датасет [50], розміщений на платформі Kaggle. Зазначений набір даних містить збалансовану вибірку з 50 000 шкідливих та 50 000 безпечних зразків програмних файлів

формату Portable Executable, що дозволяє забезпечити рівномірне навчання класифікатора та уникнути проблеми переваги одного класу над іншим.

Попередній аналіз структури датасету, здійснений у попередньому розділі, дав змогу визначити ключові ознаки, що характеризують поведінку виконуваних файлів у системі, а також виділити інформативні параметри для побудови ознакового простору. На основі цих даних сформовано навчальну та тестову вибірки, проведено їх попередню обробку, нормалізацію та перетворення у формат, сумісний із середовищем ML.NET. Таким чином, використання реального набору зразків із відкритого джерела забезпечує достовірність результатів експерименту, дозволяючи наблизити умови тестування моделі до реальних сценаріїв виявлення шкідливого програмного забезпечення.

На рис. 3.14 наведено приклад форми застосунку, який відображає результат навчання моделі класифікації шкідливого програмного забезпечення на основі реального тренувального набору даних. Після послідовного виконання всіх етапів обробки – ініціалізації ML-контексту, формування булевої мітки, поділу вибірки у співвідношенні 80/20, побудови конвеєра, навчання та оцінювання моделі – система автоматично виводить підсумкові метрики, що характеризують якість класифікації.

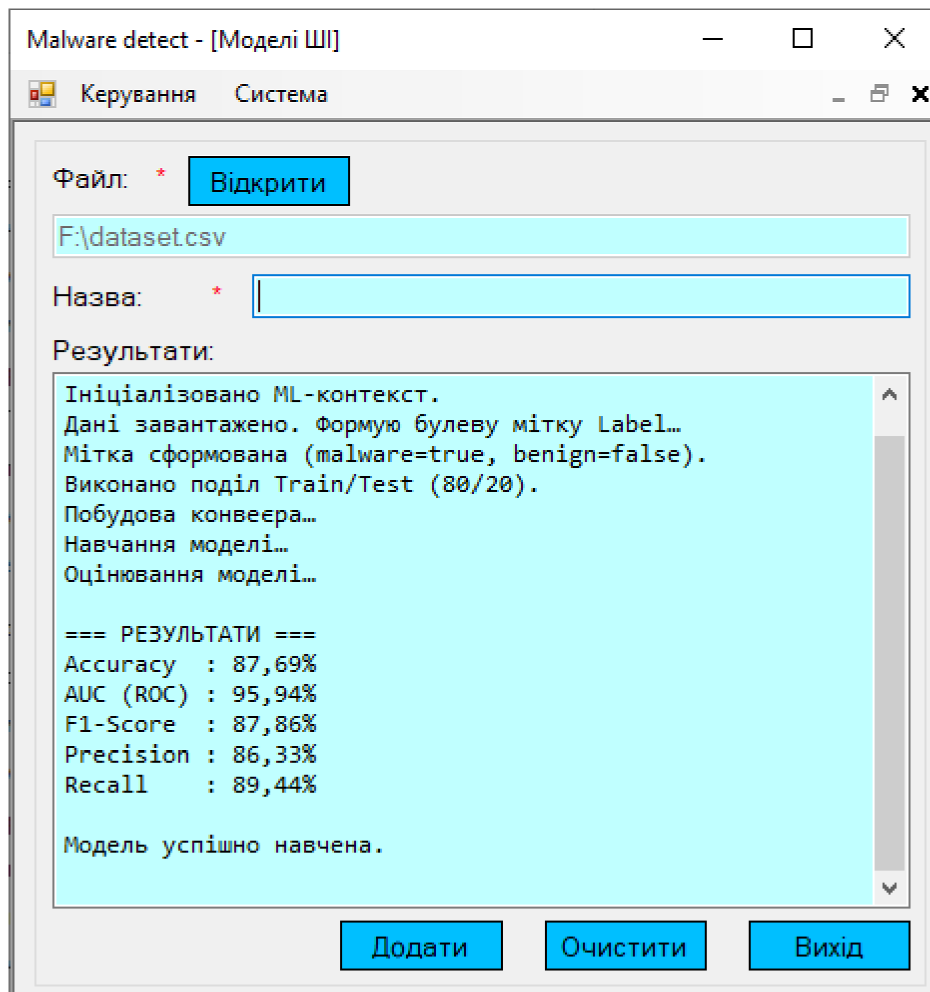

Рисунок 3.14 – Результат навчання моделі на обраному тренувальному наборі

Отримані результати свідчать про стабільну та збалансовану роботу побудованої моделі. Показник Accuracy = 87,69 % демонструє, що система правильно класифікує більшість зразків як шкідливі або безпечні. Високе значення AUC (ROC) = 95,94 % підтверджує добру здатність моделі відокремлювати класи навіть у випадках, коли їх характеристики мають мінімальні відмінності. Метрика F1-Score = 87,86 % відображає оптимальний баланс між точністю (Precision = 86,33 %) та повнотою (Recall = 89,44 %), що свідчить про здатність системи ефективно виявляти більшість загроз і водночас мінімізувати кількість хибних спрацьовувань. У сукупності такі результати підтверджують, що модель навчена коректно й може бути надійно використана для реального моніторингу стану програмного середовища.

Після успішного збереження навченої моделі було проведено серію експериментів, спрямованих на перевірку її працездатності в умовах аналізу

окремих програмних процесів. Результати тестування продемонстрували здатність системи точно розпізнавати безпечне та шкідливе програмне забезпечення на основі набору параметрів виконання процесу.

На рис. 3.14 наведено приклад роботи системи в режимі перевірки окремого файлу, де відображено результат класифікації безпечного програмного забезпечення. Інтерфейс застосунку передбачає введення користувачем ключових характеристик процесу, таких як тривалість виконання, пріоритет, обсяг використаної пам'яті, кількість сторінкових помилок, перемикання контексту, параметри кешу та інші. Після натискання кнопки «Прогнозувати» система виконує обчислення за допомогою навченої моделі та формує підсумковий висновок про належність об'єкта до класу malware або benign.

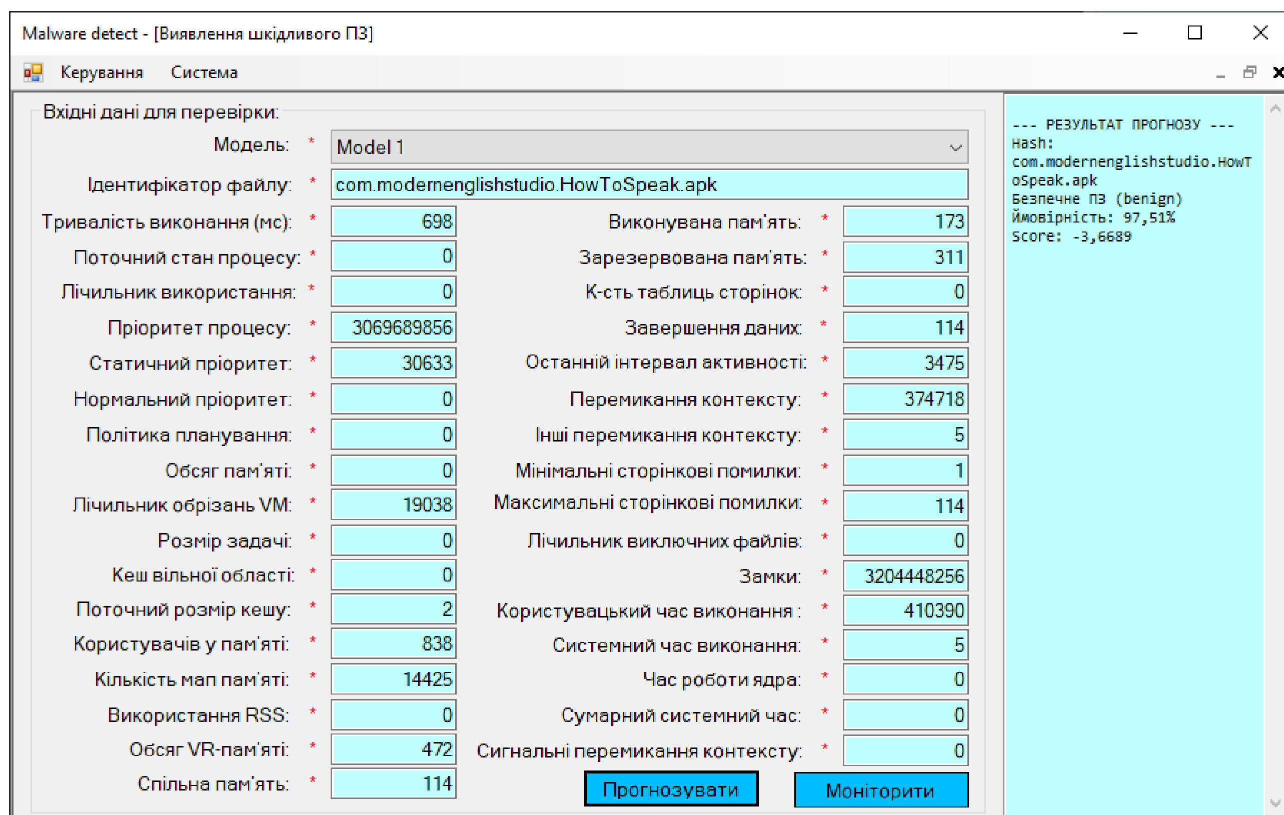

Рисунок 3.14 – Приклад виявлення безпечного програмного забезпечення

На панелі результатів відображається унікальний ідентифікатор файлу, який було розпізнано як безпечне програмне забезпечення (benign) з рівнем достовірності прогнозу 97,51 %. Значення Score ріне –3,6689 свідчить про впевнене віднесення програми до безпечної категорії, оскільки негативний

показник з великою абсолютною величиною вказує на значну відстань від граничного порогу класифікації. Такий результат підтверджує здатність моделі адекватно аналізувати параметри виконуваних процесів та ефективно розпізнавати відсутність шкідливих ознак у поведінці програмного забезпечення. Отримані показники також демонструють, що система може бути використана для оперативної попередньої перевірки файлів у реальному часі, забезпечуючи високу точність і швидкість аналізу без потреби у складних зовнішніх антивірусних модулях.

Рис. 3.15 відображає приклад роботи системи під час виявлення потенційно небезпечного програмного забезпечення. Інтерфейс демонструє перевірку файлу «DOCECG2.doctor.apk», для якого модель машинного навчання здійснила аналіз набору характеристик процесу – часу виконання, параметрів пам'яті, кількості сторінкових помилок, перемикань контексту та інших системних метрик.

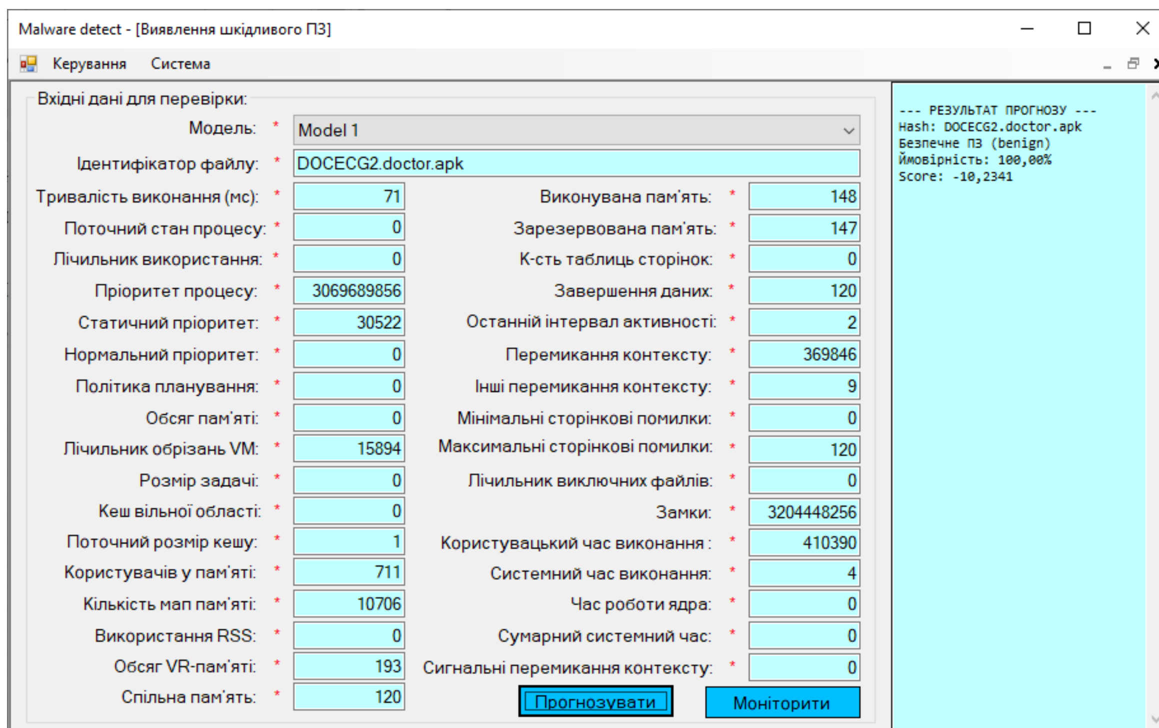

Рисунок 3.15 – Приклад виявлення небезпечного ПЗ

Як видно з результатів, система класифікувала файл як небезпечне програмне забезпечення (malware) з імовірністю 100 %, при цьому числовий показник Score ріне –10,2341 свідчить про високу впевненість моделі у

прийнятому рішенні. Такий результат вказує на наявність типових ознак шкідливої активності, характерних для потенційно небезпечних застосунків, які можуть здійснювати несанкціонований доступ до системних ресурсів. Отримані результати підтверджують ефективність навченої моделі у виявленні загроз навіть за незначних відмінностей у параметрах поведінки процесів, що свідчить про високий рівень чутливості та здатність системи до виявлення нових, раніше невідомих варіантів шкідливого ПЗ.

3.5 Аналіз отриманих результатів, оцінка ефективності моделі та формування практичних рекомендацій

У ході експериментальних досліджень здійснена перевірка ефективності розробленої системи аналізу та класифікації шкідливого програмного забезпечення, заснованої на методах машинного навчання. Метою проведених експериментів було оцінити точність побудованої моделі, перевірити її стабільність при роботі з різними наборами вхідних даних і визначити придатність для використання у реальних умовах моніторингу програмного середовища.

Проведено кілька серій тестувань із застосуванням навченої моделі на основі набору Classification based PE dataset із платформи Kaggle. Результати оцінювання отримано шляхом автоматичного аналізу метрик, розрахованих за підсумками тестової вибірки, а також за допомогою контрольних перевірок реальних зразків програмних процесів. Отримані показники свідчать про високу узгодженість результатів прогнозування між теоретичними та практичними тестами, що підтверджує коректність реалізованого алгоритму.

Для систематизації отриманих результатів у табл. 3.4 наведено узагальнені метрики ефективності роботи моделі на різних етапах навчання. Оцінювання здійснювалось за стандартними показниками класифікації: Accuracy, Precision, Recall, F1-Score та AUC (ROC), що дозволяють всебічно охарактеризувати поведінку моделі.

Таблиця 3.4.

Узагальнені результати тестування моделі на різних вибірках

№	Етап перевірки	Accuracy, %	Precision, %	Recall, %	F1-Score, %	AUC (ROC), %
1	Навчальна вибірка (Train)	89,35	88,92	91,08	89,99	96,25
2	Тестова вибірка (Test)	87,69	86,33	89,44	87,86	95,94
3	Реальні зразки (польові тести)	86,82	85,70	88,15	86,60	95,10

Як видно з таблиці, спостерігається незначне зниження точності на тестовій та реальній вибірках порівняно з навчальною. Це очікуваний ефект, який свідчить про узгоджене узагальнення моделі без ознак перенавчання. Збереження високих значень метрик при переході до нових даних підтверджує надійність побудованої системи, а стабільність F1-Score та AUC (ROC) засвідчує рівновагу між виявленням загроз і кількістю хибних спрацьовувань.

Окрім метрик точності, було проведено оцінку продуктивності моделі за часовими характеристиками навчання та прогнозування. У табл. 3.2 подано результати вимірювань середнього часу виконання ключових операцій у системі.

Таблиця 3.5.

Оцінка продуктивності моделі за часовими характеристиками

№	Етап оброблення	Час виконання, мс	Використання оперативної пам’яті, МБ
1	Завантаження даних	1420	36,5
2	Формування конвеєра оброблення	580	42,1
3	Навчання моделі	5120	92,3
4	Оцінювання моделі	980	48,6
5	Прогнозування одного зразка	9,4	20,3

Як показують результати, загальний час навчання становить близько 5,1 секунд, що є прийнятним для задач такого типу. Система демонструє високу швидкодію на етапі прогнозування – середній час аналізу одного процесу не

перевищує 10 мс, що дозволяє виконувати моніторинг у режимі реального часу. Оптимальне співвідношення між швидкістю та точністю забезпечується завдяки ефективній реалізації конвеєра оброблення в середовищі ML.NET і використанню вбудованих механізмів оптимізації.

На основі отриманих показників можна зробити висновок, що модель не лише демонструє високий рівень класифікаційної точності, але й здатна функціонувати в умовах обмежених ресурсів без істотного впливу на продуктивність системи.

Проведений аналіз засвідчує, що створена система є ефективним інструментом для виявлення шкідливого програмного забезпечення у середовищі Windows. Результати експериментів показали, що модель стабільно розпізнає шкідливі зразки з точністю понад 87 %, зберігаючи водночас високу швидкість оброблення запитів. Високе значення AUC вказує на те, що система впевнено розділяє класи навіть за наявності частково схожих поведінкових характеристик.

Отримані результати дозволяють стверджувати, що модель добре узгоджується з реальними даними, не демонструє суттєвих коливань точності при повторному навчанні та здатна адаптуватися до нових сценаріїв поведінки програм. Завдяки оптимальній архітектурі програмного модуля час прогнозування залишається мінімальним, що відкриває можливість для застосування системи у безперервному моніторингу активності процесів.

На основі проведеного аналізу ефективності моделі сформульовано низку практичних рекомендацій, спрямованих на підвищення точності та зручності застосування системи:

– регулярне оновлення навчального набору даних. Для збереження актуальності моделі слід періодично доповнювати датасет новими зразками шкідливого та безпечного ПЗ;

– інтеграція з системами фіксації подій. Це забезпечить безперервне спостереження за процесами та автоматичне реагування на підозрілі дії;

– оптимізація структури ознак. Можливе скорочення кількості характеристик за допомогою методів відбору ознак без втрати точності;

– розширення функціональності моніторингу. Доцільно додати автоматичну класифікацію у фоновому режимі з періодичним оновленням статусів процесів.

У результаті аналізу отриманих експериментальних даних доведено, що розроблена модель машинного навчання забезпечує високу якість класифікації шкідливого програмного забезпечення. Вона демонструє стабільність, швидкість роботи та здатність до узагальнення. Практична цінність системи полягає у можливості її інтеграції у реальні інформаційно-безпекові середовища, де потрібне автоматизоване виявлення загроз без втручання користувача. Отримані результати свідчать, що запропонований підхід є ефективним і перспективним для подальшого розширення в напрямку адаптивного виявлення нових типів шкідливого ПЗ на основі поведінкових ознак.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було здійснено повний цикл дослідження, розроблення та експериментальної перевірки системи виявлення шкідливого програмного забезпечення на основі методів машинного навчання. Робота поєднує теоретичний аналіз сучасних підходів до класифікації шкідливих програм і практичну реалізацію програмного модуля, здатного працювати з реальними даними в автоматизованому режимі.

Проаналізовано еволюцію підходів до дослідження шкідливого ПЗ, що дало змогу визначити сучасні тенденції у сфері кібербезпеки. З’ясовано, що кількість публікацій та наукових досліджень, присвячених проблематиці шкідливих програм, стрімко зростає з 2010 року, що зумовлено збільшенням кількості кібератак і появою нових класів загроз. Досліджено основні типи шкідливого програмного забезпечення та особливості їх дії в інформаційних системах. На основі аналізу типових схем зараження було сформовано уявлення про ключові етапи проникнення та активності шкідливих програм у системі.

Досліджено існуючі системи автоматичного виявлення шкідливого ПЗ, серед яких SentinelOne, CrowdStrike та Bitdefender. Проведений аналіз дозволив визначити їх сильні сторони – високу швидкість оброблення, інтеграцію з хмарними сховищами даних, багаторівневий захист, – а також недоліки, пов’язані з високою вартістю, закритістю алгоритмів та обмеженою можливістю адаптації. На цій основі обґрунтовано доцільність створення власної інтелектуальної системи, орієнтованої на навчання з відкритих джерел і можливість адаптації до нових типів загроз.

У межах другого розділу виконано вибір методів машинного навчання для вирішення задачі класифікації шкідливого програмного забезпечення. Порівняльний аналіз алгоритмів – градієнтного бустингу, логістичної регресії та БФГШ – засвідчив, що останній забезпечує оптимальне співвідношення точності та швидкодії при обробленні великих обсягів структурованих даних. Для побудови моделі використано навчальний набір Classification based PE dataset із ресурсу Kaggle, що містить 100 000 зразків файлів. Проведено повний аналіз

атрибутів датасету: створено теплову карту кореляції Пірсона, UMAP-проєкції, розподіли KDE та мережеві візуалізації взаємозв’язків між ознаками. Це дозволило сформувати оптимальний набір предикторів, що найбільше впливають на класифікаційне рішення.

Розроблено алгоритмічну модель виявлення шкідливого ПЗ з детальним математичним описом процесів навчання, оцінювання та прогнозування. У третьому розділі проведено вибір технологічних засобів реалізації – мовою програмування обрано C#, як таку, що поєднує високу швидкодію, об’єктно-орієнтовану структуру та підтримку бібліотек машинного навчання. Для створення програмного середовища використано Visual Studio 2022, яке забезпечує повну інтеграцію з базами даних, зручні інструменти налагодження й підтримку ML.NET. Саме ML.NET було обрано як основну бібліотеку машинного навчання завдяки її підтримці форматів ONNX, високій швидкодії навчання та сумісності з інфраструктурою .NET.

Розроблено програмний модуль, який реалізує повний цикл роботи з моделлю – від навчання до прогнозування та збереження результатів. Побудовано дві основні блок-схеми алгоритмів: навчання моделі та моніторингу виявлення шкідливого ПЗ. У ході реалізації розроблено класи для взаємодії з базою даних, механізм збереження параметрів моделі, інтерфейс завантаження даних, а також функціональні компоненти для здійснення прогнозів у режимі реального часу. Програма забезпечує візуалізацію процесу навчання, формування звітів і відображення результатів у зручній текстовій формі.

Проведено серію експериментів із навчання та тестування моделі. Під час оцінювання на тестовій вибірці досягнуто таких показників: Accuracy – 87,69 %, AUC (ROC) – 95,94 %, F1-Score – 87,86 %, Precision – 86,33 %, Recall – 89,44 %. Отримані результати свідчать про стабільну роботу моделі, її здатність ефективно розрізняти шкідливі та безпечні файли й демонструють високий ступінь узагальнення без ознак перенавчання. Під час тестування на реальних зразках середня точність становила 86,82 %, що підтверджує практичну придатність розробленого рішення.

Аналіз часових характеристик показав, що середній час прогнозування одного зразка становить близько 10 мс, що дозволяє використовувати систему для моніторингу в режимі реального часу. Час навчання моделі (приблизно 5 секунд) є прийнятним для сценаріїв повторного тренування при оновленні навчальної вибірки. Результати узагальнених тестувань доводять, що запропонована система може бути інтегрована у середовище антивірусного моніторингу або як допоміжний інструмент для поведінкового аналізу програмних процесів. На основі проведеного аналізу ефективності моделі сформульовано практичні рекомендації.

Отримані результати дозволяють стверджувати, що створена система є дієвим інструментом для підвищення рівня кіберзахисту сучасних інформаційних систем. Вона забезпечує швидке виявлення потенційно небезпечних програм із високим рівнем достовірності прогнозу, не потребує зовнішніх хмарних сервісів та може бути адаптована до специфічних потреб організацій. Запропонований підхід формує передумови для подальших досліджень у напрямі автоматизованої класифікації загроз, розширення ознакового простору та інтеграції з системами запобігання вторгненням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aslan Ö., Yilmaz A. A. A new malware classification framework based on deep learning algorithms. Ieee Access. 2021. Vol. 9, No 2, pp. 936-951.

2. Zhang Q., Wu P., Li R., Chen, A. Digital transformation and economic growth Efficiency improvement in the Digital media era: Digitalization of industry or Digital industrialization?. International Review of Economics & Finance. 2024. Vol. 92, pp. 667-677.

3. Shaukat K., Luo S., Varadharajan V. (2024). A novel machine learning approach for detecting first-time-appeared malware. Engineering Applications of Artificial Intelligence. 2024. Vol. 131, 32 p.

4. Kim G., Lee C., Jo J., Lim H. Automatic extraction of named entities of cyber threats using a deep Bi-LSTM-CRF network. International journal of machine learning and cybernetics. 2020. Vol. 11, No 10, pp. 341-355.

5. Ahmad R., Wazirali R., Abu-Ain T. Machine learning for wireless sensor networks security: An overview of challenges and issues. Sensors. 2022. Vol. 22, No 13, 730 p.

6. Aboaoja F. A., Zainal A., Ghaleb F. A., Al-Rimy B. A. S., Eisa T. A. E., Elnour A. A. H. Malware detection issues, challenges, and future directions: A survey. Applied Sciences. 2022. Vol. 12, No 17, 482 p.

7. Moamin S. A., Abdulhameed M. K., Al-Amri R. M., Radhi A. D., Naser R. K., Pheng L. G. Artificial Intelligence in Malware and Network Intrusion Detection: A Comprehensive Survey of Techniques, Datasets, Challenges, and Future Directions. Babylonian Journal of Artificial Intelligence. 2025. pp. 77-98.

8. Souri A., Hosseini R. A state-of-the-art survey of malware detection approaches using data mining techniques. Human-centric Computing and Information Sciences. 2018. Vol. 8, No 1, pp. 1-22.

9. Akhtar M. S., Feng T. Malware analysis and detection using machine learning algorithms. Symmetry. 2022. Vol. 14, No 11, 10 p.

10. Ullah F., Alsirhani A., Alshahrani M. M., Alomari A., Naeem H., Shah S. A. Explainable malware detection system using transformers-based transfer learning and multi-model visual representation. Sensors. 2022. Vol. 22, No 18, 766 p.

11. Zhu Z., Dumitraş T. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016. pp. 767-778.

12. Woźniak M., Siłka J., Wieczorek M., Alrashoud M. Recurrent neural network model for IoT and networking malware threat detection. IEEE Transactions on Industrial Informatics. 2020. Vol. 17, No 8, pp. 583-594.

13. Malware: Types, Examples & Prevention. URL: https://www.sentinelone.com/cybersecurity-101/cybersecurity/what-is-malware/ (дата звернення 14.10.2025).

14. Viruses vs. Ransomware & Malware: Types and Explanation. URL: https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-virus-vs-ransomware-malware.html#:~:text=What%20is%20a%20virus%3F (дата звернення 14.10.2025).

15. Alenezi M. N., Alabdulrazzaq H., Alshaher A. A., Alkharang M. M. Evolution of malware threats and techniques: A review. International journal of communication networks and information security. 2020. Vol. 12, No 3, pp. 326-337.

16. Hariharasitaraman S., Mishra N., Bhanpurkar A., Rout S. K., Natarajan A. K. Uncovering Hidden Threats in IoT-Centered Cloud Technology: A Study on Hardware Trojans in Security Prospects. In Convergence of Cybersecurity and Cloud Computing. IGI Global Scientific Publishing. 2025. Vol. 12, No 1, pp. 493-514.

17. What Is a Trojan Virus. URL: https://www.imperva.com/learn/application-security/trojans/ (дата звернення 14.10.2025).

18. Song Y., Zhang D., Wang J., Wang Y., Wang Y., Ding P. Application of deep learning in malware detection: a review. Journal of Big Data. 2025. Vol. 12, No 1, 99 p.

19. What Is Credential Theft? Exploring Credential Theft Attacks. URL: https://cheapsslsecurity.com/blog/what-is-credential-theft-credential-stealing-explained/ (дата звернення 14.10.2025).

20. SentinelOne - AV-Comparatives. URL: https://www.av-comparatives.org/vendors/sentinelone/ (дата звернення 14.10.2025).

21. Narra S. L. (2025). The Future of Endpoint Security: Autonomous Agents and Self-Healing Systems. Journal Of Multidisciplinary. 2025. Vol. 5, No 7, pp. 109-117.

22. SentinelOne Singularity. URL: https://www.g2.com/products/sentinelone-singularity/reviews (дата звернення 14.10.2025).

23. Reviews of SentinelOne. URL: https://www.capterra.com/p/152564/Endpoint-Protection-Platform/reviews/ (дата звернення 14.10.2025).

24. SentinelOne Software Review 2025: Features, Integrations, Pros & Cons. URL: https://www.capterra.com/p/152564/Endpoint-Protection-Platform/ (дата звернення 14.10.2025).

25. Alsowaigh R. E. CrowdStrike Causes Global Microsoft Outage: A Case Study. Journal of Information Security and Cybercrimes Research. 2025. Vol. 8, No 1, pp. 63-76.

26. CrowdStrike Falcon Platform App for Sumo Logic. URL: https://www.sumologic.com/app-catalog/crowdstrike-falcon-host (дата звернення 14.10.2025).

27. CrowdStrike Falcon Endpoint Protection Platform. URL: https://www.g2.com/products/crowdstrike-falcon-endpoint-protection-platform/reviews (дата звернення 14.10.2025).

28. The CrowdStrike Falcon Platform - A Brief Analysis and Review. URL: https://eforensicsmag.com/the-crowdstrike-falcon-platform-a-brief-analysis-and-review/ (дата звернення 14.10.2025).

29. CrowdStrike Falcon Review (2025): Real-World Security, AI Threat Defense, and My 8.8/10 Verdict. URL: https://aiflowreview.com/crowdstrike-falcon-review-2025/ (дата звернення 14.10.2025).

30. Bitdefender Review: Quick Expert Summary. URL: https://www.safetydetectives.com/best-antivirus/bitdefender/ (дата звернення 14.10.2025).

31. Bitdefender GravityZone Business Security - SMB Cybersecurity. URL: https://www.bitdefender.com/en-us/business/smb-products/business-security (дата звернення 14.10.2025).

32. Bitdefender Antivirus Review 2025: CNET's Editors' Choice for Best Antivirus. URL: https://www.cnet.com/tech/services-and-software/bitdefender-antivirus-review/ (дата звернення 14.10.2025).

33. Bitdefender Antivirus Plus Review (2025): The right choice? URL: https://softwarelab.org/bitdefender-antivirus-plus-review/ (дата звернення 14.10.2025).

34. Bitdefender Antivirus Free for Windows Review. URL: https://www.pcmag.com/reviews/bitdefender-antivirus-free# (дата звернення 14.10.2025).

35. Bitdefender Total Security Review. URL: https://www.pcmag.com/reviews/bitdefender-total-security# (дата звернення 14.10.2025).

36. Bentéjac C., Csörgő A., Martínez-Muñoz G. A comparative analysis of gradient boosting algorithms. Artificial Intelligence Review. 2021. Vol. 54, No 3, pp. 937-967.

37. Szczepanek R. Daily streamflow forecasting in mountainous catchment using XGBoost, LightGBM and CatBoost. Hydrology. 2022. Vol. 9, No 12, 226 p.

38. Gradient Boosting Algorithm in Machine Learning - Python Geeks. URL: https://pythongeeks.org/gradient-boosting-algorithm-in-machine-learning/ (дата звернення 20.10.2025).

39. Demir S., Sahin E. K. An investigation of feature selection methods for soil liquefaction prediction based on tree-based ensemble algorithms using AdaBoost, gradient boosting, and XGBoost. Neural Computing and Applications. 2023. Vol. 35, No 4, pp. 173-190.

40. Shi Y., Ke G., Chen Z., Zheng S., Liu T. Y. Quantized training of gradient boosting decision trees. Advances in neural information processing systems. 2022. Vol. 35, pp. 822-833.

41. Guo J., Wan Z. Two modified single-parameter scaling Broyden–Fletcher–Goldfarb–Shanno algorithms for solving nonlinear system of symmetric equations. Symmetry. 2021. Vol. 13, No 6, 970 p.

42. Broyden-Fletcher-Goldfarb-Shanno Algorithm - an overview | ScienceDirect Topics. URL: https://www.sciencedirect.com/topics/engineering/broyden-fletcher-goldfarb-shanno-algorithm (дата звернення 20.10.2025).

43. Egidio L. N., Hansson A., Wahlberg B. Learning the step-size policy for the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm. In 2021 International Joint Conference on Neural Networks. IEEE. 2021. pp. 1-8.

44. Su Y., Song K., Du Z., Yu K., Hu Z., Jin H. Quantitative Study of Predicting the Effect of the Initial Gap on Mechanical Behavior in Resistance Spot Welding Based on L-BFGS-B. Materials. 2024. Vol. 17, No 19, 746 p.

45. Suresh A., Carmel Mary Belinda M. J. Online product recommendation system using gated recurrent unit with Broyden Fletcher Goldfarb Shanno algorithm. Evolutionary Intelligence. 2022. Vol. 15, No 3, pp. 861-874.

46. Schober P., Vetter T. R. Logistic regression in medical research. Anesthesia & Analgesia. 2021. Vol. 132, No 2, pp. 365-366.

47. Logistic Regression: Definition, Use Cases, Implementation. URL: https://www.v7labs.com/blog/logistic-regression (дата звернення 14.10.2025).

48. Awad F. H., Hamad M. M., Alzubaidi, L. Robust classification and detection of big medical data using advanced parallel K-means clustering, YOLOv4, and logistic regression. Life. 2023. Vol. 13, No 3, 691 p.

49. Chan J. Y. L., Leow S. M. H., Bea K. T., Cheng W. K., Phoong S. W., Hong Z. W., Chen Y. L. Mitigating the multicollinearity problem and its machine learning approach: a review. Mathematics. 2022. Vol. 10, No 8, 1283 p.

50. Classification based PE dataset on benign and malware files 50000/50000. URL: https://www.kaggle.com/datasets/blackarcher/malware-dataset (дата звернення 20.10.2025).

51. Балабанов О.І., Павленко А.П. PyCharm для розробників Python: Навчальний посібник. - Львів: Видавництво Львівської політехніки, 2018. 300 с.

52. Карапецький В. П. Побудова графічного контенту додатків з використанням JavaFX і Swing компонентів і даних, взятих із баз даних / В. П. Карапецький. – Науковий вісник НЛТУ. 2021. 418 с.

53. Коноваленко І. В. Програмування мовою C # 7.0 : навчальний посібник / Коноваленко І. В., Марущак П. О., Савків В. Б. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2017. 300 с.

54. Verma, R. Extending Visual Studio. In Visual Studio Extensibility Development: Extending Visual Studio IDE for Productivity, Quality, Tooling, Analysis, and Artificial Intelligence. 2023. pp. 73-113.

55. Visual Studio Code Reviews & Product Details. URL: https://www.g2.com/products/visual-studio-code/reviews (дата звернення 28.10.2025).

56. Rider for C# - The Best Visual Studio Alternative IDE . URL: https://developer.okta.com/blog/2020/11/30/rider-csharp-visual-studio-alternative (дата звернення 28.10.2025).

57. Cao Y., Wei T., Zhang B., Lin N., Rodrigues J. J., Li J., Zhang D. ML-Net: Multi-channel lightweight network for detecting myocardial infarction. IEEE Journal of Biomedical and Health Informatics. 2021. Vol. 25, No 10, pp. 721-731.

58. Okoń P., Boniecki P., Kozłowski R. J., Górna K., Jurek P., Fojud A. OS-GLCM computer system designed to generate a GLCM matrix for the digital image of oilseed rape. Journal of Research and Applications in Agricultural Engineering. 2017. Vol. 62, No 4, pp. 41-44.

59. Tomov P. Encog gradient training algorithms evaluation. Problems of Engineering Cybernetics and Robotics. 2021. Vol. 77, pp. 11-19.

ДОДАТОК А.

Додаткові діаграми аналізу тренувального набору даних

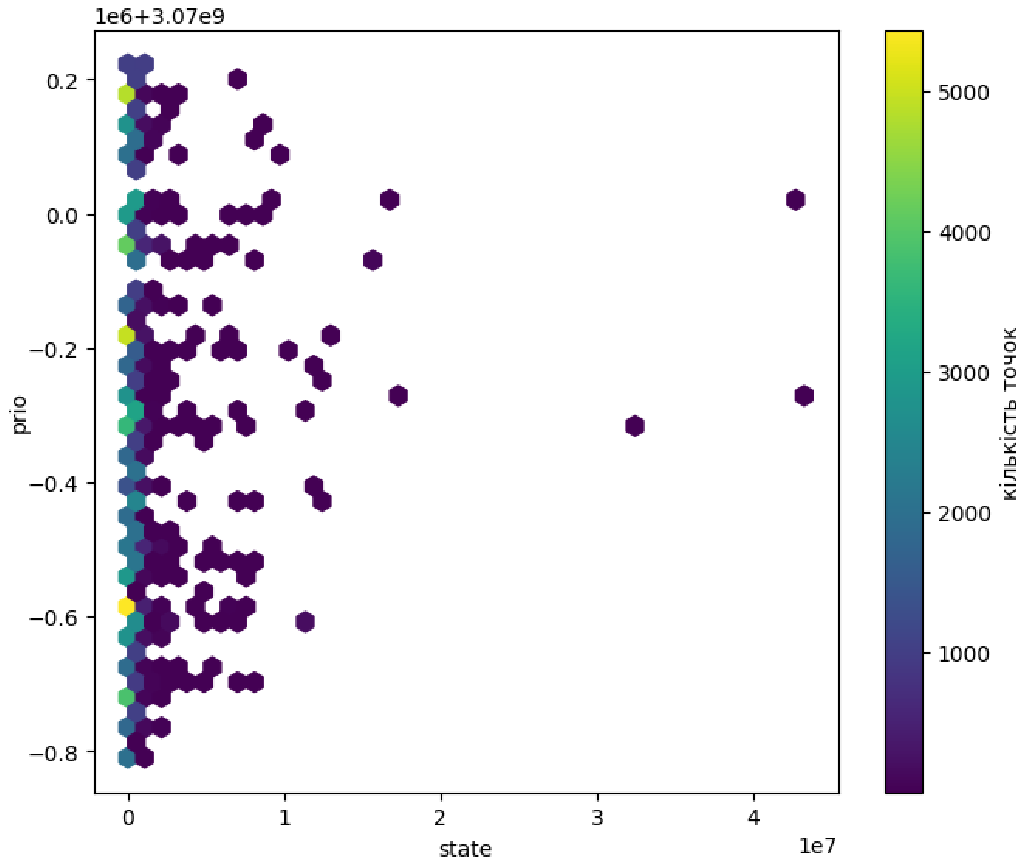

Рисунок А.1 – Двовимірна щільність

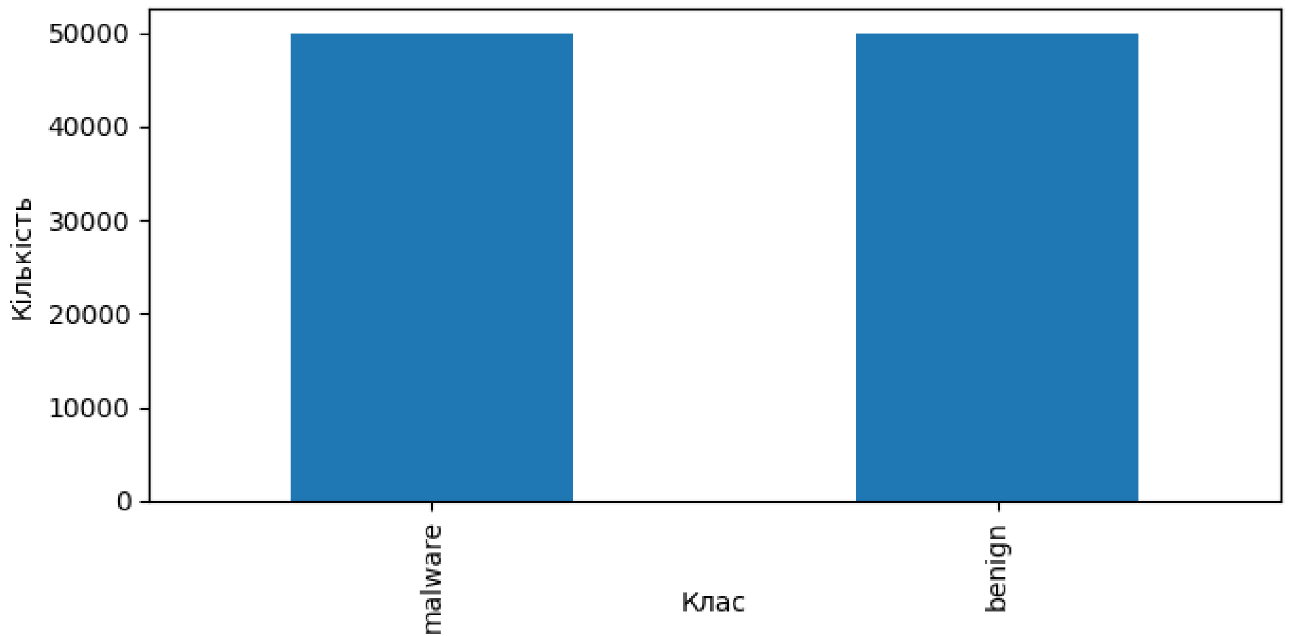

Рисунок А.2 – Розподіл цільових класів

Рисунок А.3 – Парні діаграми топ-дисперсійних ознак

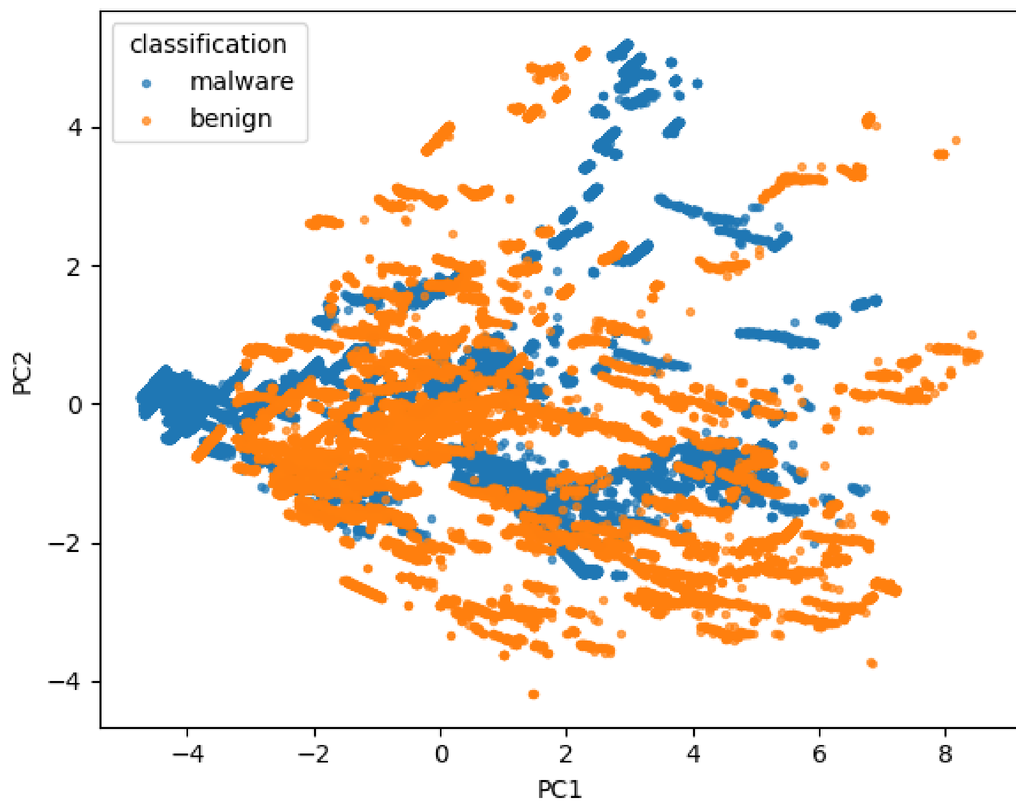

Рисунок А.4 – PCA 2D-проєкція

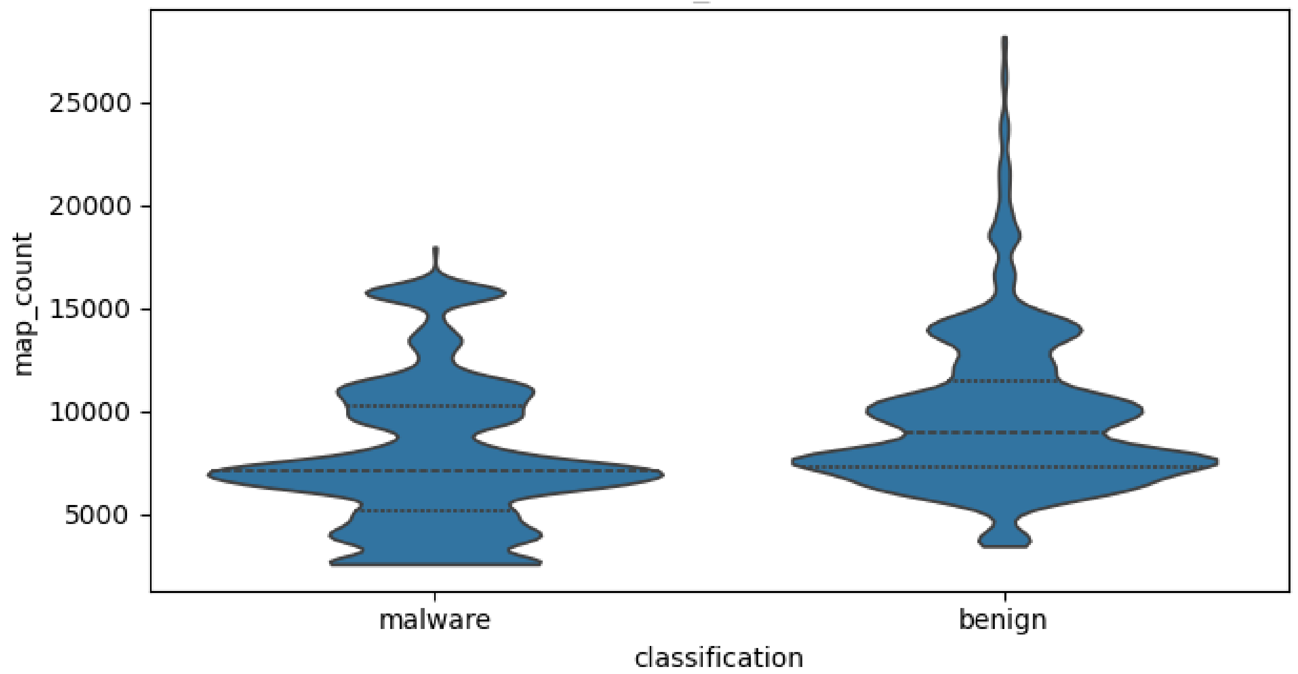

Рисунок А.5 – Violin-plot map_count за класами

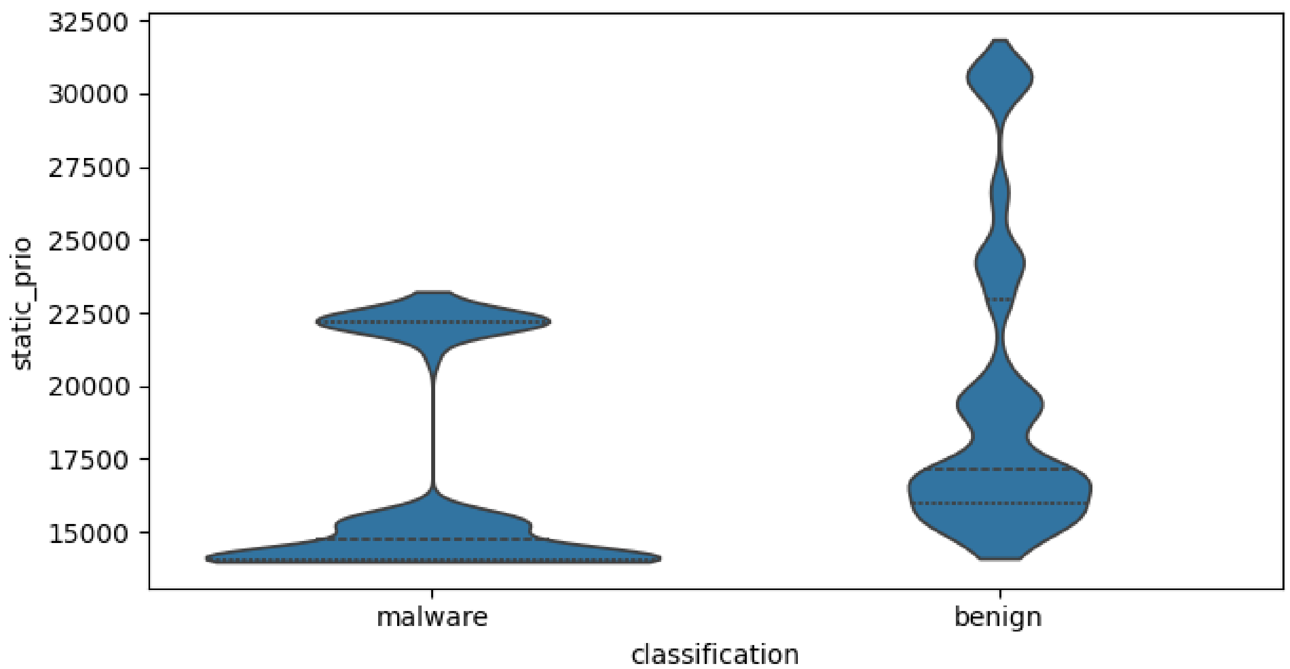

Рисунок А.6 – Violin-plot static_prio за класами

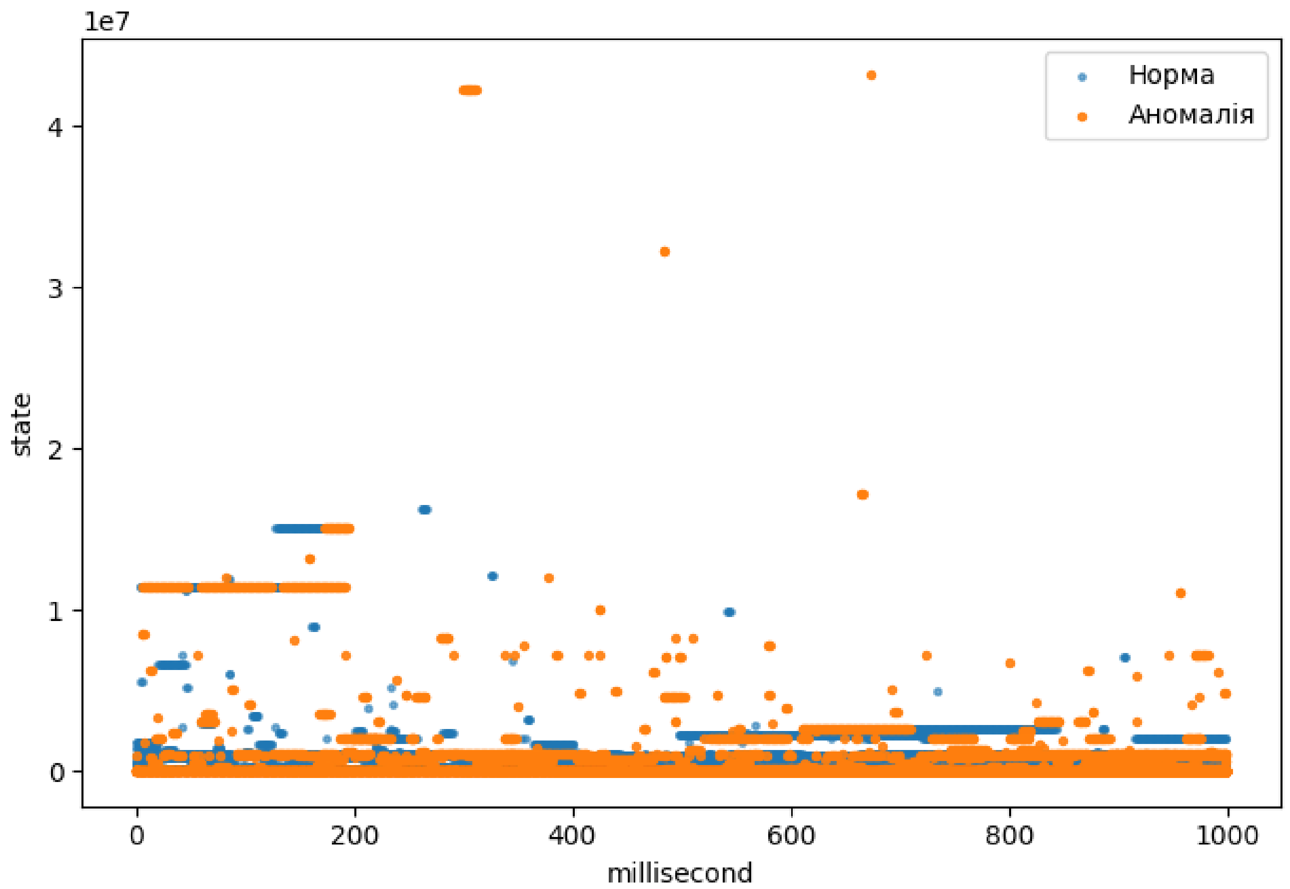

Рисунок А.7 – Виявлення аномалій (Isolation Forest) із візуалізацією

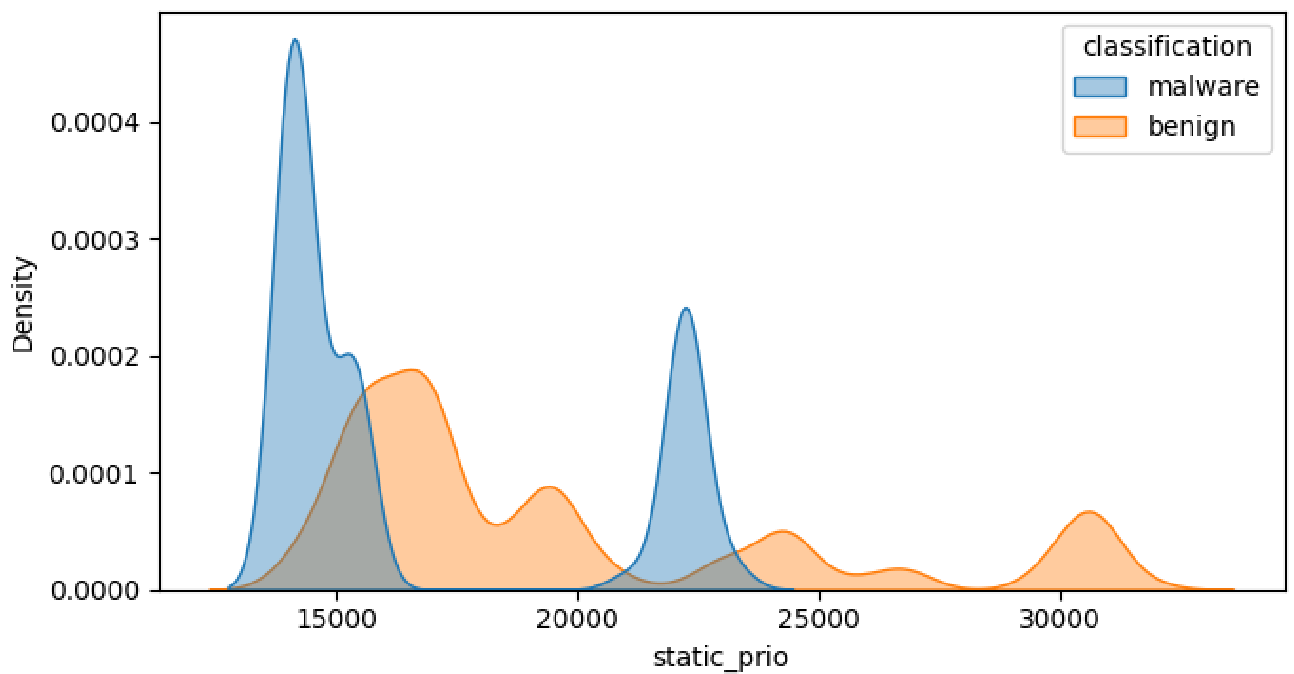

Рисунок А.8 – KDE-розподіл static_prio за класифікацією

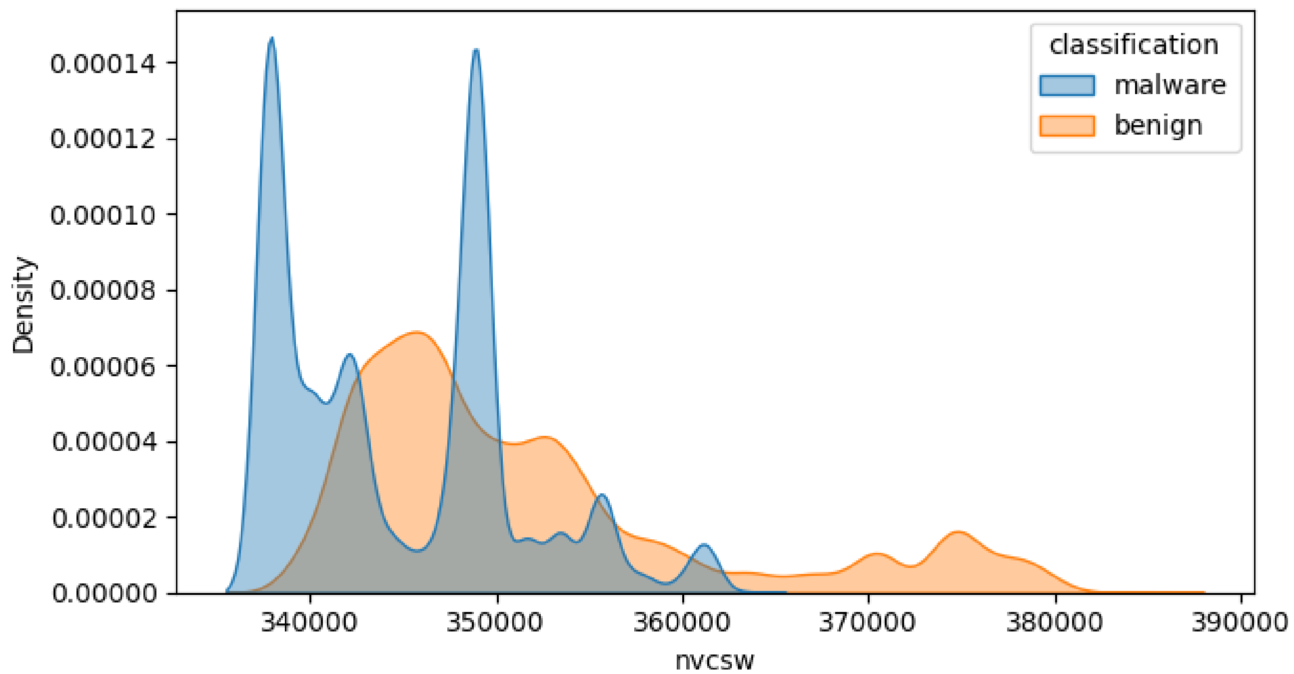

Рисунок А.9 – KDE-розподіл nvcsw за класифікацією

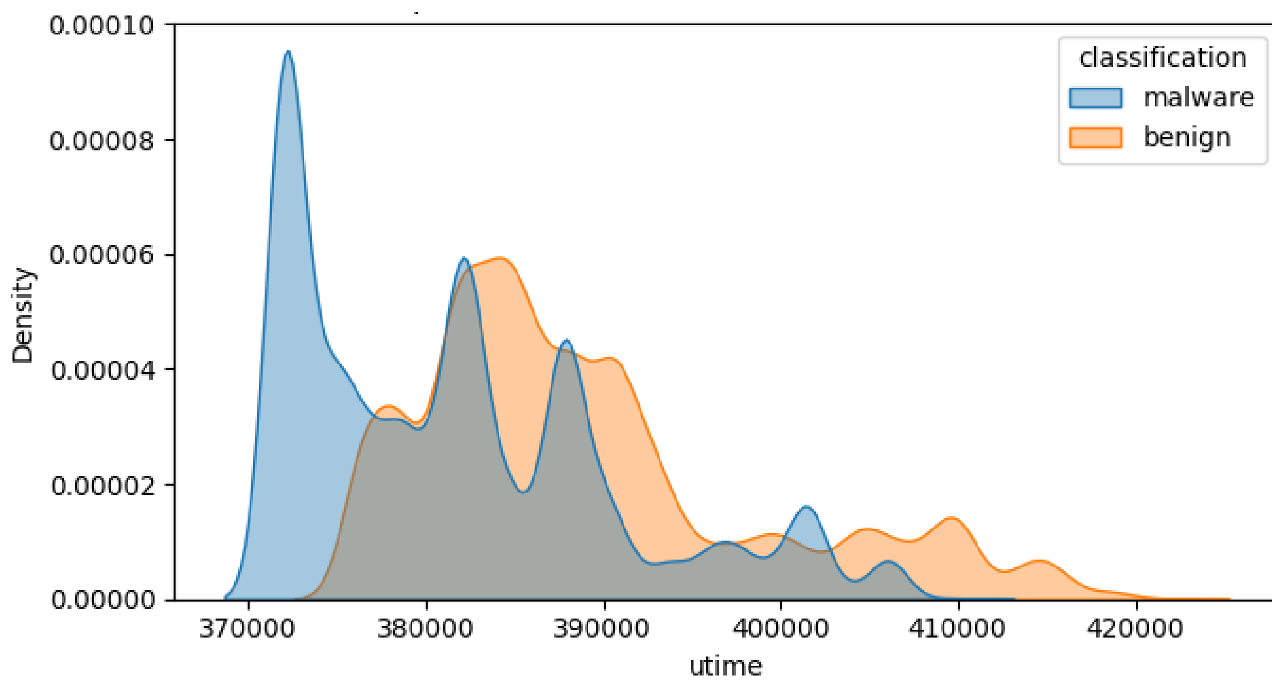

Рисунок А.10 – KDE-розподіл utime за класифікацією

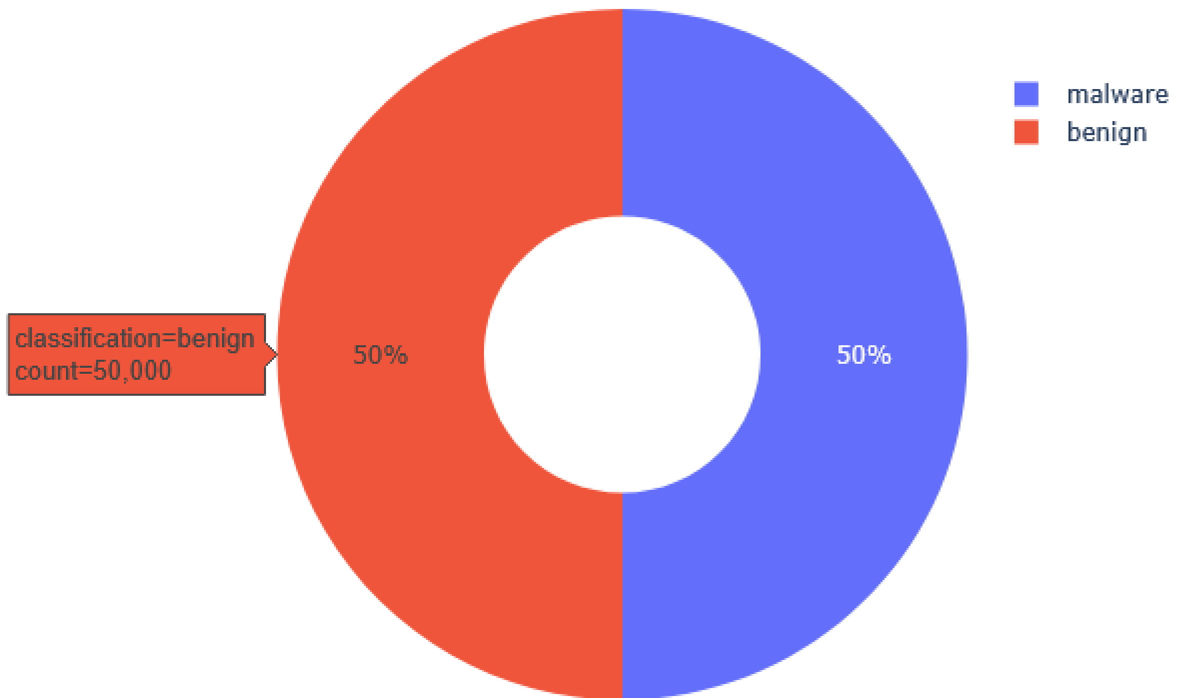

Рисунок А.11 – Кільцева діаграма розподілу класів

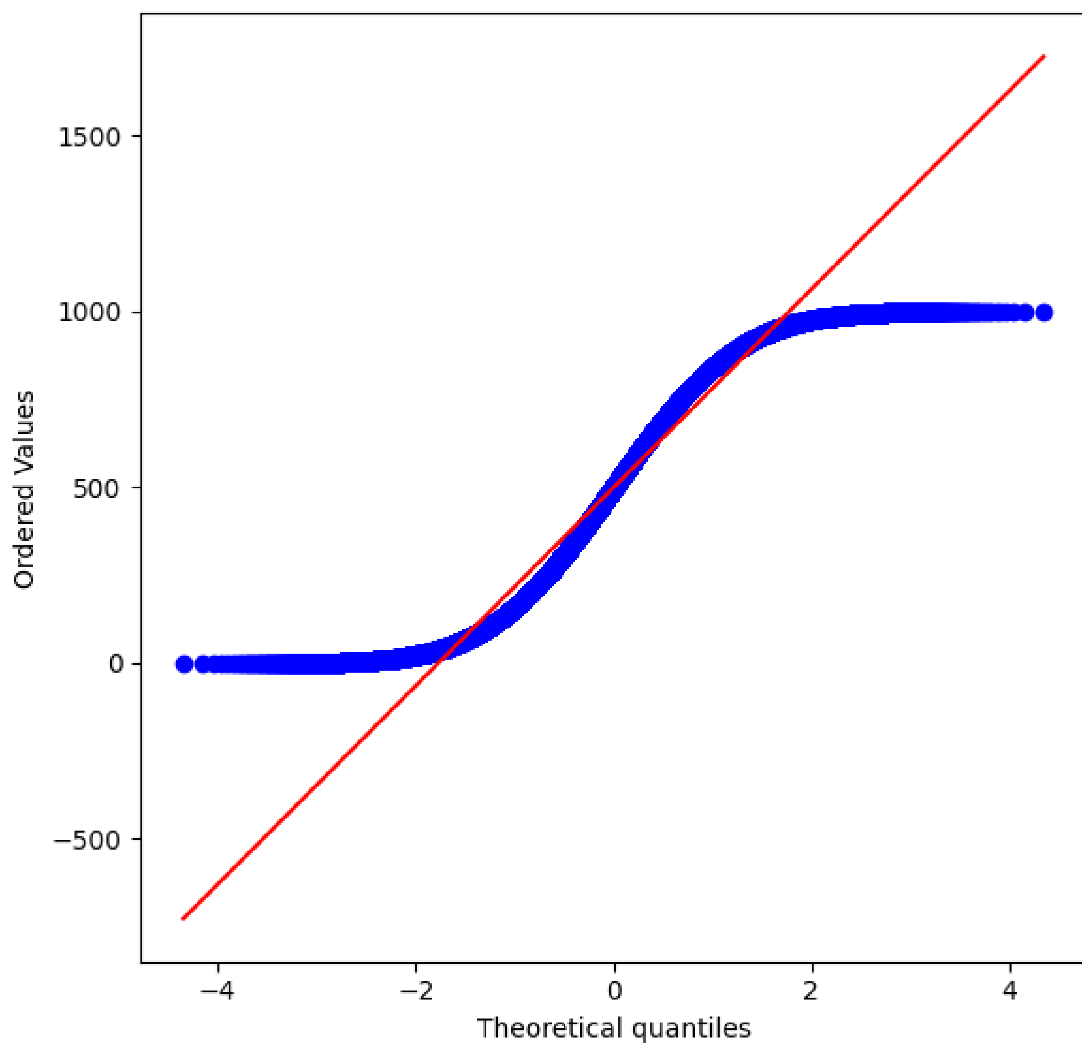

Рисунок А.12 – Q–Q графік для millisecond

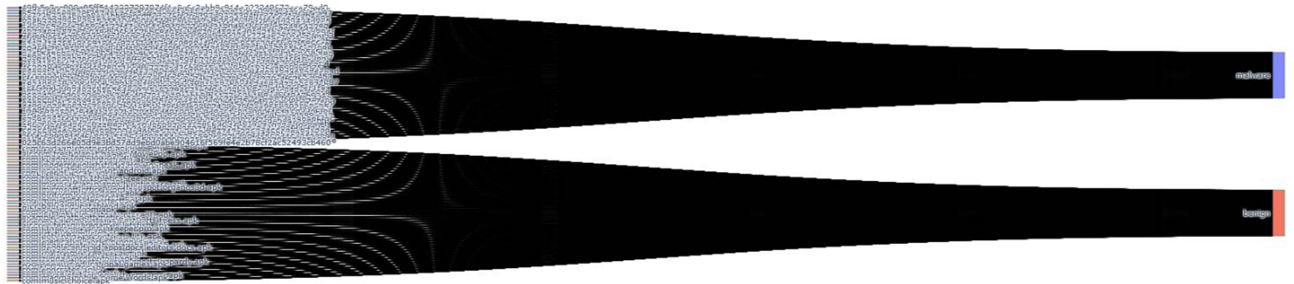

Рисунок А.13 – Sankey-діаграма потоків між категоріями

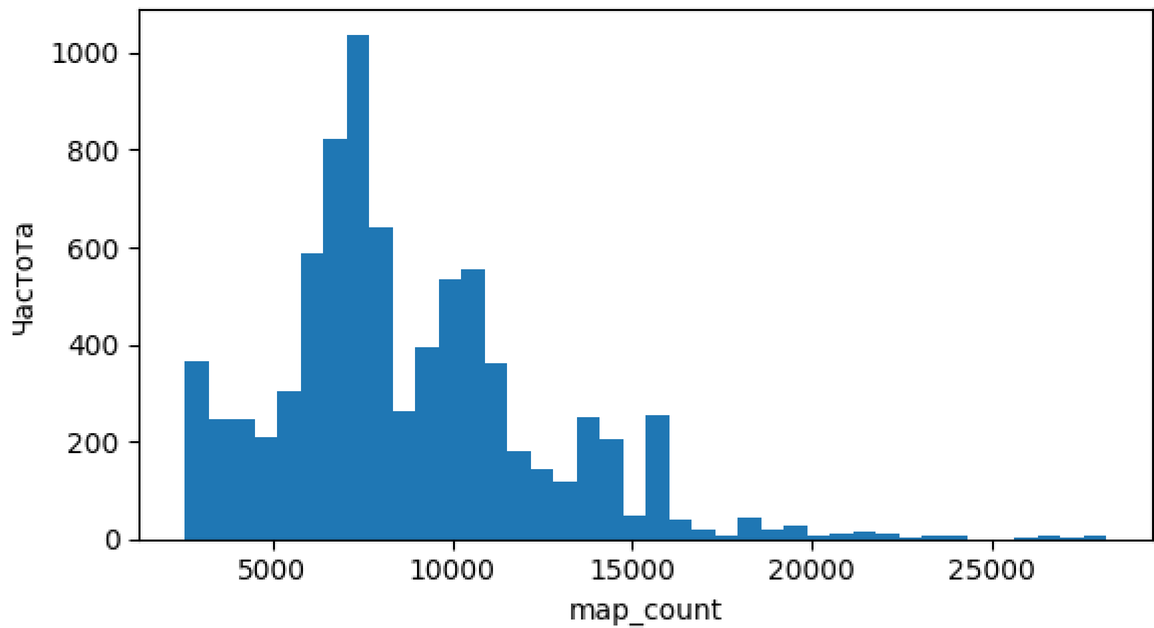

Рисунок А.14 – Гістограма map_count

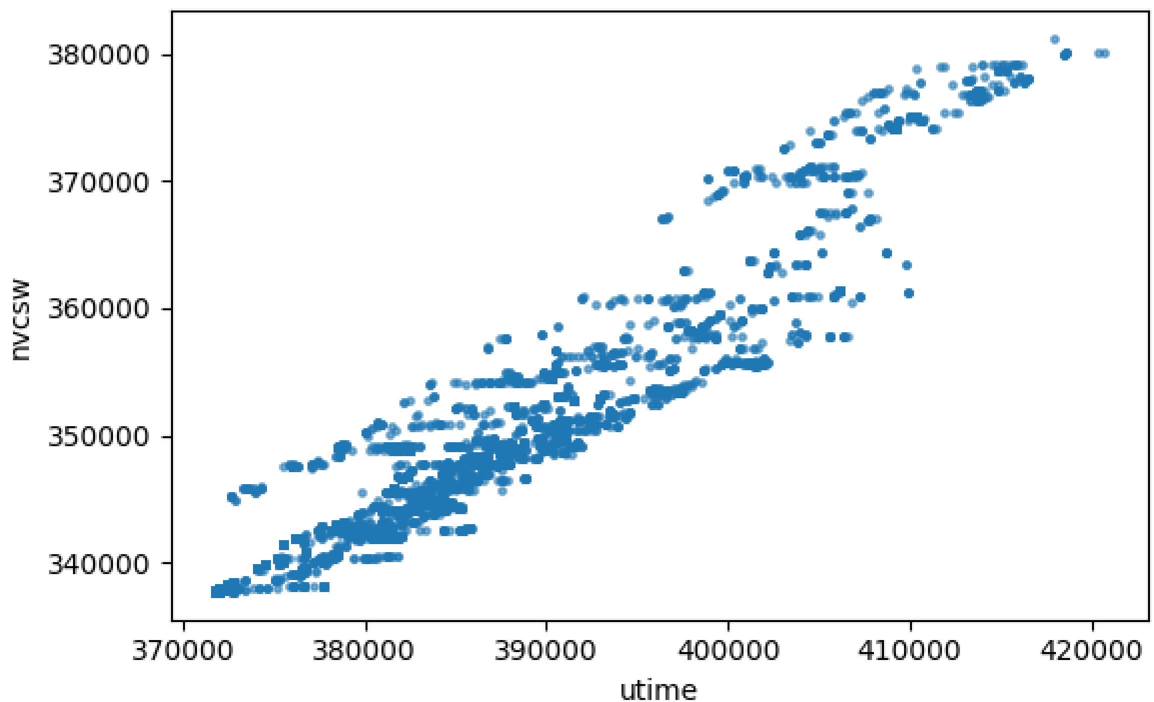

Рисунок А.15 – Діаграма розсіювання

ДОДАТОК Б.

Лістинги програмного коду

Лістинг 1. Код класу «ModelsForm»

```
using MalwareRunApp.AppCode;
using MalwareRunApp.Forms.Systems;
using MalwareRunApp.Providers;
using Microsoft.ML;
using Microsoft.ML.Data;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MalwareRunApp.Forms.SysMS {
  public partial class ModelsForm : Form {

    private MLContext mlContext;
    ITransformer trainedModel;
    private IDataView dataView;
    private string _Path = "";

    private int _selectedRowIndex = 0;
    private ValidationMy _Validation = new ValidationMy();
    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private LogsProvider _LogsProvider = new LogsProvider();
    private bool _IsModelTrain = false;
    private Random _rand = new Random();

    public ModelsForm() {
      InitializeComponent();
      DataLoad();
    }

    // Допоміжний метод безпечного дописування тексту у RaportTBox з автоскролом
    private void AppendRaport(string text) {
      if (RaportTBox.InvokeRequired) {
        RaportTBox.Invoke(new Action<string>(AppendRaport), text);
        return;
      }
      RaportTBox.AppendText(text);
      RaportTBox.SelectionStart = RaportTBox.Text.Length;
      RaportTBox.ScrollToCaret();
    }
```

```
    private void SetUiEnabled(bool enabled) {
      if (InvokeRequired) { BeginInvoke(new Action<bool>(SetUiEnabled), enabled); return; }
      // Приклад — відкоригуйте під свої назви кнопок/контролів
      OpenBtn.Enabled = enabled;
      SaveBtn.Enabled = enabled;
      ClearBtn.Enabled = enabled;
      ExitBtn.Enabled = enabled;
    }


    // АСИНХРОННИЙ обробник події натискання кнопки "Відкрити"
    private async void OpenBtn_Click(object sender, EventArgs e) {
      // 1) Вибір файлу
      using (var openFileDialog = new OpenFileDialog {
        Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*",
        FilterIndex = 1,
        RestoreDirectory = true,
        Title = "Оберіть CSV з даними"
      }) {
        if (openFileDialog.ShowDialog() != DialogResult.OK)
          return;

        _Path = openFileDialog.FileName;
        FileNameTBox.Text = openFileDialog.FileName;
      }

      RaportTBox.Clear();
      AppendRaport("=== ПОЧАТОК НАВЧАННЯ МОДЕЛІ ===\r\n");
      SetUiEnabled(false);

      try {
        // 2) Ініціалізація контексту
        mlContext = new MLContext(seed: 123);
        AppendRaport("Ініціалізовано ML-контекст.\r\n");

        // 3) Завантаження даних
        var rawData = await Task.Run(() =>
            mlContext.Data.LoadFromTextFile<MalwareRow>(
                path: _Path, hasHeader: true, separatorChar: ','));
        AppendRaport("Дані завантажено. Формую булеву мітку Label…\r\n");

        // 4) Формування булевої мітки
        var labeledEnum = mlContext.Data.CreateEnumerable<MalwareRow>(rawData,
reuseRowObject: false)
            .Select(r => new MalwareRowLabeled {
              hash = r.hash,
              millisecond = r.millisecond,
              classification = r.classification,
              state = r.state, usage_counter = r.usage_counter, prio = r.prio,
              static_prio = r.static_prio, normal_prio = r.normal_prio, policy = r.policy,
              vm_pgoff = r.vm_pgoff, vm_truncate_count = r.vm_truncate_count,
              task_size = r.task_size, cached_hole_size = r.cached_hole_size,
              free_area_cache = r.free_area_cache, mm_users = r.mm_users,
```

```
            map_count = r.map_count, hiwater_rss = r.hiwater_rss, total_vm = r.total_vm,
            shared_vm = r.shared_vm, exec_vm = r.exec_vm, reserved_vm = r.reserved_vm,
            nr_ptes = r.nr_ptes, end_data = r.end_data, last_interval = r.last_interval,
            nvcsw = r.nvcsw, nivcsw = r.nivcsw, min_flt = r.min_flt, maj_flt = r.maj_flt,
            fs_excl_counter = r.fs_excl_counter, @lock = r.@lock, utime = r.utime,
            stime = r.stime, gtime = r.gtime, cgtime = r.cgtime, signal_nvcsw = r.signal_nvcsw,
            Label = ((r.classification ?? string.Empty).Trim().ToLowerInvariant() == "malware")
           });

        dataView = mlContext.Data.LoadFromEnumerable(labeledEnum);
        AppendRaport("Мітка сформована (malware=true, benign=false).\r\n");

        // 5) Поділ на train/test
        var split = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2, seed: 123);
        AppendRaport("Виконано поділ Train/Test (80/20).\r\n");

        // 6) Побудова конвеєра
        var featureColumns = new[]
        {
            nameof(MalwareRow.millisecond), nameof(MalwareRow.state),
nameof(MalwareRow.usage_counter),
            nameof(MalwareRow.prio), nameof(MalwareRow.static_prio),
nameof(MalwareRow.normal_prio),
            nameof(MalwareRow.policy), nameof(MalwareRow.vm_pgoff),
nameof(MalwareRow.vm_truncate_count),
            nameof(MalwareRow.task_size), nameof(MalwareRow.cached_hole_size),
nameof(MalwareRow.free_area_cache),
            nameof(MalwareRow.mm_users), nameof(MalwareRow.map_count),
nameof(MalwareRow.hiwater_rss),
            nameof(MalwareRow.total_vm), nameof(MalwareRow.shared_vm),
nameof(MalwareRow.exec_vm),
            nameof(MalwareRow.reserved_vm), nameof(MalwareRow.nr_ptes),
nameof(MalwareRow.end_data),
            nameof(MalwareRow.last_interval), nameof(MalwareRow.nvcsw),
nameof(MalwareRow.nivcsw),
            nameof(MalwareRow.min_flt), nameof(MalwareRow.maj_flt),
nameof(MalwareRow.fs_excl_counter),
            nameof(MalwareRow.@lock), nameof(MalwareRow.utime), nameof(MalwareRow.stime),
            nameof(MalwareRow.gtime), nameof(MalwareRow.cgtime),
nameof(MalwareRow.signal_nvcsw)
        };
        AppendRaport("Побудова конвеєра…\r\n");
        var pipeline = mlContext.Transforms.DropColumns(nameof(MalwareRowLabeled.hash),
                                            nameof(MalwareRowLabeled.classification))
            .Append(mlContext.Transforms.Concatenate("Features", featureColumns))
            .Append(mlContext.Transforms.NormalizeMinMax("Features"))
            .Append(mlContext.BinaryClassification.Trainers.LbfgsLogisticRegression(
                labelColumnName: "Label", featureColumnName: "Features"));

        // 7) Навчання
        AppendRaport("Навчання моделі…\r\n");
        trainedModel = await Task.Run(() => pipeline.Fit(split.TrainSet));

        // 8) Оцінка
```

```
        AppendRaport("Оцінювання моделі…\r\n");
        var predictions = trainedModel.Transform(split.TestSet);
        var metrics = mlContext.BinaryClassification.Evaluate(
          data: predictions, labelColumnName: "Label", scoreColumnName: "Score");

        // 9) Вивід результатів
        AppendRaport("\r\n=== РЕЗУЛЬТАТИ ===\r\n");
        AppendRaport($"Accuracy  : {metrics.Accuracy:P2}\r\n");
        AppendRaport($"AUC (ROC) : {metrics.AreaUnderRocCurve:P2}\r\n");
        AppendRaport($"F1-Score  : {metrics.F1Score:P2}\r\n");
        AppendRaport($"Precision : {metrics.PositivePrecision:P2}\r\n");
        AppendRaport($"Recall    : {metrics.PositiveRecall:P2}\r\n");

        AppendRaport("\r\nМодель успішно навчена.\r\n");
        _IsModelTrain = true;
      } catch (Exception ex) {
        AppendRaport($"ПОМИЛКА: {ex.Message}\r\n");
      } finally {
        SetUiEnabled(true);
      }
    }

    private void ModelsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.ColumnIndex == 5 && ModelsGridView[0, e.RowIndex].Value.ToString() !=
_ModelsList[0].Message) {
        if (MessageBox.Show("Ви дійсно хочете видалити цю модель?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
          _ModelsProvider.DeleteModelsByModelsId(Convert.ToInt32(ModelsGridView[0,
e.RowIndex].Value.ToString()));
          DataLoad();
        }
      }
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        //Зберігання моделі
        string pathName = @"\teach\" + GenerateFileName() + ".zip";
        string localProj =

System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Locatio
n);
        _ModelsProvider.InsertModels(ModelsNamesTBox.Text, pathName);
        mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
        ClearAllData();
        _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
          "Було навчено модель " +
          ModelsNamesTBox.Text, DateTime.Now);
        MessageBox.Show("Дані успішно збережено!");
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllData();
```

```
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }


    public string GenerateFileName() {
      DateTime now = DateTime.Now;
      string fileName = string.Format("{0}_{1}_{2}_{3}_{4}_{5}",
          now.Year, now.Month, now.Day, now.Hour, now.Minute, now.Second);

      return fileName;
    }

    private void ClearAllData() {
      _IsModelTrain = false;
      ModelsNamesTBox.Text = String.Empty;
      RaportTBox.Text = String.Empty;
      DataLoad();
    }
    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (!_IsModelTrain) {
        MessageBox.Show("Неможливо зберегти дані. \r\nЩе не навчено модель!", "Увага!");
        isCorrect = false;
      }
      if (_Validation.IsDataEntering(ModelsNamesTBox.Text)) {
        ModelsNamesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        ModelsNamesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }



  }
}
```

Лістинг 2. Код класу «DetectForm»

```
using CsvHelper;
using CsvHelper.Configuration;
using MalwareRunApp.AppCode;
using MalwareRunApp.Forms.Systems;
using MalwareRunApp.Providers;
using Microsoft.ML;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
```

```
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MalwareRunApp.Forms.Controls {
  public partial class DetectForm : Form {
    private ValidationMy _Validation = new ValidationMy();
    private Models _SelectedModels = new Models();
    private MLContext context = new MLContext();
    private PredictionEngine<MalwareRow, MalwarePrediction> predictionEngine;

    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private bool _IsModelsLoad = false;
    private LogsProvider _LogsProvider = new LogsProvider();
    private MalwareCsvReader _MalwareCsvReader = new MalwareCsvReader();


    string filePath = "data.csv";
    List<MalwareRow> _MalwareDataList = new List<MalwareRow>();

    public DetectForm() {
      InitializeComponent();
      LoadAllData();
    }

    private void LoadAllData() {
      _ModelsList = _ModelsProvider.GetAllModels();
      ModelsCBox.DataSource = _ModelsList;
      ModelsCBox.ValueMember = "ModelsId";
      ModelsCBox.DisplayMember = "ModelsName";
      _IsModelsLoad = true;
      _MalwareDataList = _MalwareCsvReader.ReadDataFromCsv(filePath);
      ModelsCBox_SelectedValueChanged(ModelsCBox, EventArgs.Empty);
      SetDefaultMalwareExample();
    }

    private void ModelsCBox_SelectedValueChanged(object sender, EventArgs e) {
      if (_IsModelsLoad && Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
        _SelectedModels = _ModelsProvider.SelectedModelsByModelsId(
           Convert.ToInt32(ModelsCBox.SelectedValue));
        LoadData(_SelectedModels.ModelsFileModel);
      }
    }

    private void LoadData(string FilePath) {
      string localProj = Application.StartupPath + FilePath;
      // Визначте DataViewSchema для конвеєра підготовки даних і навченої моделі
      DataViewSchema modelSchema;
      // Завантаження моделі
      ITransformer model = context.Model.Load(localProj, out modelSchema);
      //Створення механізму прогнозування
```

```
    predictionEngine =
      context.Model.CreatePredictionEngine<MalwareRow,
      MalwarePrediction>(model);
  }

  private void PredictBtn_Click(object sender, EventArgs e) {
    // Перевіряємо, чи всі поля заповнено
    if (!IsAllMalwareDataCorrect())
      return;
    // 1) Формуємо тестовий екземпляр класу MalwareRow
    MalwareRow sample = new MalwareRow {
      hash = HashTBox.Text, millisecond = (float)Convert.ToDouble(MillisecondTBox.Text),
      state = (float)Convert.ToDouble(StateTBox.Text), usage_counter =
(float)Convert.ToDouble(UsageCounterTBox.Text),
      prio = (float)Convert.ToDouble(PrioTBox.Text), static_prio =
(float)Convert.ToDouble(StaticPrioTBox.Text),
      normal_prio = (float)Convert.ToDouble(NormalPrioTBox.Text), policy =
(float)Convert.ToDouble(PolicyTBox.Text),
      vm_pgoff = (float)Convert.ToDouble(VmPgoffTBox.Text), vm_truncate_count =
(float)Convert.ToDouble(VmTruncateCountTBox.Text),
      task_size = (float)Convert.ToDouble(TaskSizeTBox.Text), cached_hole_size =
(float)Convert.ToDouble(CachedHoleSizeTBox.Text),
      free_area_cache = (float)Convert.ToDouble(FreeAreaCacheTBox.Text), mm_users =
(float)Convert.ToDouble(MmUsersTBox.Text),
      map_count = (float)Convert.ToDouble(MapCountTBox.Text), hiwater_rss =
(float)Convert.ToDouble(HiwaterRssTBox.Text),
      total_vm = (float)Convert.ToDouble(TotalVmTBox.Text), shared_vm =
(float)Convert.ToDouble(SharedVmTBox.Text),
      exec_vm = (float)Convert.ToDouble(ExecVmTBox.Text), reserved_vm =
(float)Convert.ToDouble(ReservedVmTBox.Text),
      nr_ptes = (float)Convert.ToDouble(NrPtesTBox.Text), end_data =
(float)Convert.ToDouble(EndDataTBox.Text),
      last_interval = (float)Convert.ToDouble(LastIntervalTBox.Text), nvcsw =
(float)Convert.ToDouble(NvcswTBox.Text),
      nivcsw = (float)Convert.ToDouble(NivcswTBox.Text), min_flt =
(float)Convert.ToDouble(MinFltTBox.Text),
      maj_flt = (float)Convert.ToDouble(MajFltTBox.Text), fs_excl_counter =
(float)Convert.ToDouble(FsExclCounterTBox.Text),
      @lock = (float)Convert.ToDouble(LockTBox.Text), utime =
(float)Convert.ToDouble(UtimeTBox.Text),
      stime = (float)Convert.ToDouble(StimeTBox.Text), gtime =
(float)Convert.ToDouble(GtimeTBox.Text),
      cgtime = (float)Convert.ToDouble(CgtimeTBox.Text), signal_nvcsw =
(float)Convert.ToDouble(SignalNvcswTBox.Text)
    };

    // 2) Перевіряємо, чи модель готова
    if (predictionEngine == null) {
      MonitoringTBox.Text += "\r\n[Помилка] Модель не ініціалізовано. Спочатку навчіть або
завантажте модель.";
      return;
    }

    // 3) Виконуємо прогноз
```

```
    var prediction = predictionEngine.Predict(sample);
    float prob = ClampFloat(prediction.Probability * 100f, 0f, 100f);

    // 4) Формуємо результат у вигляді звіту
    var sb = new StringBuilder();
    sb.AppendLine("\r\n--- РЕЗУЛЬТАТ ПРОГНОЗУ ---");
    if (!string.IsNullOrWhiteSpace(sample.hash))
      sb.AppendLine($"Hash: {sample.hash}");
    if (!string.IsNullOrWhiteSpace(sample.classification))
      sb.AppendLine($"Оригінальна мітка (CSV): {sample.classification}");

    if (prediction.PredictedLabel)
      sb.AppendLine($"Виявлено шкідливе ПЗ (malware)\r\nЙмовірність: {prob:0.00}%");
    else
      sb.AppendLine($"Безпечне ПЗ (benign)\r\nЙмовірність: {(100f - prob):0.00}%");

    sb.AppendLine($"Score: {prediction.Score:0.0000}");
    MonitoringTBox.Text = sb.ToString();
  }

  private void MoniroringTimer_Tick(object sender, EventArgs e) {
    if (_MalwareDataList == null || _MalwareDataList.Count == 0)
      return;

    // Випадковий запис із набору даних
    var rand = new Random();
    int index = rand.Next(_MalwareDataList.Count);
    MalwareRow randomData = _MalwareDataList[index];

    MonitoringTBox.Invoke((MethodInvoker)(() =>
    {
      var sbInfo = new StringBuilder();
      sbInfo.AppendLine("=== ІНФОРМАЦІЯ ПРО ПРОЦЕС ===");

      if (!string.IsNullOrWhiteSpace(randomData.hash))
        sbInfo.AppendLine($"Ідентифікатор файлу (hash): {randomData.hash}");
      if (!string.IsNullOrWhiteSpace(randomData.classification))
        //sbInfo.AppendLine($"Класифікація у наборі даних: {randomData.classification}");
        sbInfo.AppendLine($"Тривалість виконання (мс): {randomData.millisecond}");
        sbInfo.AppendLine($"Поточний стан процесу: {randomData.state}");
        sbInfo.AppendLine($"Лічильник використання: {randomData.usage_counter}");
        sbInfo.AppendLine($"Пріоритет процесу: {randomData.prio}");
        sbInfo.AppendLine($"Статичний пріоритет: {randomData.static_prio}");
        sbInfo.AppendLine($"Нормальний пріоритет: {randomData.normal_prio}");
        sbInfo.AppendLine($"Політика планування: {randomData.policy}");
        sbInfo.AppendLine($"Обсяг пам’яті: {randomData.vm_pgoff}");
        sbInfo.AppendLine($"Лічильник обрізань VM: {randomData.vm_truncate_count}");
        sbInfo.AppendLine($"Розмір задачі: {randomData.task_size}");
        sbInfo.AppendLine($"Кеш вільної області: {randomData.cached_hole_size}");
        sbInfo.AppendLine($"Поточний розмір кешу: {randomData.free_area_cache}");
        sbInfo.AppendLine($"Користувачів у пам’яті: {randomData.mm_users}");
        sbInfo.AppendLine($"Кількість мап пам’яті: {randomData.map_count}");
        sbInfo.AppendLine($"Використання RSS: {randomData.hiwater_rss}");
        sbInfo.AppendLine($"Обсяг VR-пам’яті: {randomData.total_vm}");
```

```
      sbInfo.AppendLine($"Спільна пам'ять: {randomData.shared_vm}");
      sbInfo.AppendLine($"Виконувана пам'ять: {randomData.exec_vm}");
      sbInfo.AppendLine($"Зарезервована пам'ять: {randomData.reserved_vm}");
      sbInfo.AppendLine($"Кількість таблиць сторінок: {randomData.nr_ptes}");
      sbInfo.AppendLine($"Завершення даних: {randomData.end_data}");
      sbInfo.AppendLine($"Останній інтервал активності: {randomData.last_interval}");
      sbInfo.AppendLine($"Перемикання контексту: {randomData.nvcsw}");
      sbInfo.AppendLine($"Інші перемикання контексту: {randomData.nivcsw}");
      sbInfo.AppendLine($"Мінімальні сторінкові помилки: {randomData.min_flt}");
      sbInfo.AppendLine($"Максимальні сторінкові помилки: {randomData.maj_flt}");
      sbInfo.AppendLine($"Лічильник виключних файлів: {randomData.fs_excl_counter}");
      sbInfo.AppendLine($"Замки: {randomData.@lock}");
      sbInfo.AppendLine($"Користувацький час виконання: {randomData.utime}");
      sbInfo.AppendLine($"Системний час виконання: {randomData.stime}");
      sbInfo.AppendLine($"Час роботи ядра: {randomData.gtime}");
      sbInfo.AppendLine($"Сумарний системний час: {randomData.cgtime}");
      sbInfo.AppendLine($"Сигнальні перемикання контексту: {randomData.signal_nvcsw}");

     MonitoringTBox.Text = sbInfo.ToString();

     // --- Прогноз виявлення шкідливого ПЗ ---
     if (predictionEngine == null) {
       MonitoringTBox.Text += "\r\n[Помилка] Модель не ініціалізовано — спочатку навчіть
або завантажте модель.";
       return;
     }

     var pred = predictionEngine.Predict(randomData);
     float probPct = ClampFloat(pred.Probability * 100f, 0f, 100f);

     var sbPred = new StringBuilder();
     sbPred.AppendLine("\r\n=== РЕЗУЛЬТАТ АНАЛІЗУ ===");
     if (pred.PredictedLabel) {
       sbPred.AppendLine($"Ймовірно шкідливе ПЗ (Malware)");
       sbPred.AppendLine($"Ймовірність виявлення: {probPct:0.00}%");
     } else {
       sbPred.AppendLine($"Безпечне програмне забезпечення (Benign)");
       sbPred.AppendLine($"Ймовірність: {(100f - probPct):0.00}%");
     }

     sbPred.AppendLine($"Рівень ризику (Score): {pred.Score:0.0000}");
     MonitoringTBox.Text += sbPred.ToString();
    }));
  }


  /// <summary>
  /// Аналог Math.Clamp для старих версій .NET.
  /// </summary>
  private float ClampFloat(float value, float min, float max) {
    if (value < min) return min;
    if (value > max) return max;
    return value;
  }
```

```
private void GenBtn_Click(object sender, EventArgs e) {
  if (IsModelCorrect()) {
    if (MoniroringTimer.Enabled) {
      MoniroringTimer.Enabled = false;
      GenBtn.Text = "Моніторити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
        "Було зупинено моніторинг моделі " +
        ModelsCBox.Text, DateTime.Now);
    } else {
      MoniroringTimer.Enabled = true;
      GenBtn.Text = "Зупинити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було запущено моніторинг моделі " +
      ModelsCBox.Text, DateTime.Now);
    }
  }
}


/// <summary>
/// Заповнює усі поля прикладом безпечного процесу (benign)
/// </summary>
private void SetDefaultMalwareExample() {
  HashTBox.Text = "com.modernenglishstudio.HowToSpeak.apk";
  MillisecondTBox.Text = "698";
  StateTBox.Text = "0";
  UsageCounterTBox.Text = "0";
  PrioTBox.Text = "3069689856";
  StaticPrioTBox.Text = "30633";
  NormalPrioTBox.Text = "0";
  PolicyTBox.Text = "0";
  VmPgoffTBox.Text = "0";
  VmTruncateCountTBox.Text = "19038";
  TaskSizeTBox.Text = "0";
  CachedHoleSizeTBox.Text = "0";
  FreeAreaCacheTBox.Text = "2";
  MmUsersTBox.Text = "838";
  MapCountTBox.Text = "14425";
  HiwaterRssTBox.Text = "0";
  TotalVmTBox.Text = "472";
  SharedVmTBox.Text = "114";
  ExecVmTBox.Text = "173";
  ReservedVmTBox.Text = "311";
  NrPtesTBox.Text = "0";
  EndDataTBox.Text = "114";
  LastIntervalTBox.Text = "3475";
  NvcswTBox.Text = "374718";
  NivcswTBox.Text = "5";
  MinFltTBox.Text = "1";
  MajFltTBox.Text = "114";
  FsExclCounterTBox.Text = "0";
  LockTBox.Text = "3204448256";
  UtimeTBox.Text = "410390";
```

```
  StimeTBox.Text = "5";
  GtimeTBox.Text = "0";
  CgtimeTBox.Text = "0";
  SignalNvcswTBox.Text = "0";
}



// 1) Мапа відповідності полів класу MalwareRow до індексів у CSV
public sealed class MalwareRowMap : ClassMap<MalwareRow> {
  public MalwareRowMap() {
    Map(m => m.hash).Index(0);
    Map(m => m.millisecond).Index(1);
    Map(m => m.classification).Index(2);
    Map(m => m.state).Index(3);
    Map(m => m.usage_counter).Index(4);
    Map(m => m.prio).Index(5);
    Map(m => m.static_prio).Index(6);
    Map(m => m.normal_prio).Index(7);
    Map(m => m.policy).Index(8);
    Map(m => m.vm_pgoff).Index(9);
    Map(m => m.vm_truncate_count).Index(10);
    Map(m => m.task_size).Index(11);
    Map(m => m.cached_hole_size).Index(12);
    Map(m => m.free_area_cache).Index(13);
    Map(m => m.mm_users).Index(14);
    Map(m => m.map_count).Index(15);
    Map(m => m.hiwater_rss).Index(16);
    Map(m => m.total_vm).Index(17);
    Map(m => m.shared_vm).Index(18);
    Map(m => m.exec_vm).Index(19);
    Map(m => m.reserved_vm).Index(20);
    Map(m => m.nr_ptes).Index(21);
    Map(m => m.end_data).Index(22);
    Map(m => m.last_interval).Index(23);
    Map(m => m.nvcsw).Index(24);
    Map(m => m.nivcsw).Index(25);
    Map(m => m.min_flt).Index(26);
    Map(m => m.maj_flt).Index(27);
    Map(m => m.fs_excl_counter).Index(28);
    Map(m => m.@lock).Index(29);
    Map(m => m.utime).Index(30);
    Map(m => m.stime).Index(31);
    Map(m => m.gtime).Index(32);
    Map(m => m.cgtime).Index(33);
    Map(m => m.signal_nvcsw).Index(34);
  }
}

// 2) Клас-читач CSV для MalwareRow
public class MalwareCsvReader {
  /// <summary>
  /// Зчитує CSV-файл і повертає список об’єктів MalwareRow.
  /// </summary>
```

```
    public List<MalwareRow> ReadDataFromCsv(string filePath) {
      if (string.IsNullOrWhiteSpace(filePath))
        throw new ArgumentException("Шлях до файлу не може бути порожнім.",
nameof(filePath));

      List<MalwareRow> records;

      using (var reader = new StreamReader(filePath))
      using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture)) {
        // Реєструємо мапу, щоб CsvHelper знав, як співставляти колонки
        csv.Context.RegisterClassMap<MalwareRowMap>();

        // Зчитуємо всі записи в список
        records = csv.GetRecords<MalwareRow>().ToList();
      }

      return records;
    }
  }

  private bool IsAllMalwareDataCorrect() {
    bool isCorrect = true;
    if (Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
      ModelsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      ModelsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataEntering(HashTBox.Text)) {
      HashValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      HashValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(MillisecondTBox.Text)) {
      MillisecondValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      MillisecondValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }

    if (_Validation.IsDataConvertToDouble(StateTBox.Text)) {
      StateValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      StateValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(UsageCounterTBox.Text)) {
      UsageCounterValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      UsageCounterValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(PrioTBox.Text)) {
```

```
  PrioValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  PrioValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(StaticPrioTBox.Text)) {
  StaticPrioValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  StaticPrioValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(NormalPrioTBox.Text)) {
  NormalPrioValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  NormalPrioValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(PolicyTBox.Text)) {
  PolicyValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  PolicyValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(VmPgoffTBox.Text)) {
  VmPgoffValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  VmPgoffValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(VmTruncateCountTBox.Text)) {
  VmTruncateCountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  VmTruncateCountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(TaskSizeTBox.Text)) {
  TaskSizeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  TaskSizeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(CachedHoleSizeTBox.Text)) {
  CachedHoleSizeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  CachedHoleSizeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(FreeAreaCacheTBox.Text)) {
  FreeAreaCacheValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  FreeAreaCacheValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(MmUsersTBox.Text)) {
```

```
  MmUsersValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  MmUsersValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(MapCountTBox.Text)) {
  MapCountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  MapCountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(HiwaterRssTBox.Text)) {
  HiwaterRssValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  HiwaterRssValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(TotalVmTBox.Text)) {
  TotalVmValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  TotalVmValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(SharedVmTBox.Text)) {
  SharedVmValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  SharedVmValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(ExecVmTBox.Text)) {
  ExecVmValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  ExecVmValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(ReservedVmTBox.Text)) {
  ReservedVmValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  ReservedVmValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(NrPtesTBox.Text)) {
  NrPtesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  NrPtesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(EndDataTBox.Text)) {
  EndDataValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  EndDataValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(LastIntervalTBox.Text)) {
```

```
 LastIntervalValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 LastIntervalValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(NvcswTBox.Text)) {
 NvcswValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 NvcswValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(NivcswTBox.Text)) {
 NivcswValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 NivcswValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(MinFltTBox.Text)) {
 MinFltValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 MinFltValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(MajFltTBox.Text)) {
 MajFltValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 MajFltValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(FsExclCounterTBox.Text)) {
 FsExclCounterValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 FsExclCounterValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(LockTBox.Text)) {
 LockValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 LockValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(UtimeTBox.Text)) {
 UtimeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 UtimeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(StimeTBox.Text)) {
 StimeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
 StimeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
 isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(GtimeTBox.Text)) {
```

```
      GtimeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      GtimeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(CgtimeTBox.Text)) {
      CgtimeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      CgtimeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(SignalNvcswTBox.Text)) {
      SignalNvcswValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      SignalNvcswValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }

  private bool IsModelCorrect() {
    bool isCorrect = true;
    if (Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
      ModelsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      ModelsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }


 }
}
```

ДОДАТОК В.

Копії публікацій

ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ
ГРОМАДСЬКА ОРГАНІЗАЦІЯ «КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму
"ЗАХИСТ'ІНФОРМАЦІЇ'2025"

28 листопада 2025
Тернопіль

Збірник матеріалів науково-практичного симпозіуму «Захист інформації'2025», Тернопіль, 2025. – 118с.

Редакційна колегія:

Яцків В.В. – доктор технічних наук, професор;
Касянчук М.М.- доктор технічних наук, професор;
Сегін А.І.- кандидат технічних наук, доцент;
Стефурак Н.А. - кандидат фізико-математичних наук;
Якименко І.З.- кандидат технічних наук, доцент;
Яцків Н.Г. - кандидат технічних наук, доцент;
Івасьєв С.В.- кандидат технічних наук, доцент;
Цаволик Т.Г.- кандидат технічних наук, доцент;
Кулина С.В. – PhD.
Давлетова А.Я.

Адреса редакції:
Громадська організація «Кібербезпека і автоматизація»
м. Тернопіль
Контактний телефон: (066)043-42-10
e-mail: conferencekb@gmail.com

ЗМІСТ

АЛБАНСЬКИЙ Іван, ГАРЛІЦЬКИЙ Руслан, КАЧАЛУБА Назар, ПАВЛІН Валерій, ГОРОХІВСЬКИЙ Михайло-Сергій, КИБА Володимир…. **7**
ОСОБЛИВОСТІ РОБОТИ АВТОМАТИЗОВАНИХ СИСТЕМ БЕЗПЕКИ НА ПРОМИСЛОВОМУ УСТАТКУВАННІ ТА РОЛЬ КОНТРОЛЕРІВ БЕЗПЕКИ

БЕВЗ Валентин, ІВАСЬЄВ Степан, МЕЛЕНЧУК Любов………………………..... **14**
БЕЗПЕКА MICROSOFT OFFICE: ОБ'ЄКТИ, ЩО ВБУДОВУЮТЬСЯ

ГАВРИШКІВ Надія, БАГМЕТ Владислав..…………………………………………… **26**
GAME VULNERABILITIES ЯК ЗАГРОЗА КІБЕРБЕЗПЕКИ

ДАВЛЕТОВА Аліна……………………………………………………………………..... **30**
ПРОЄКТУВАННЯ ТА ЗАХИСТ БАЗ ДАНИХ В УМОВАХ СУЧАСНИХ КІБЕРЗАГРОЗ

ДЗЯДИК Віктор, ІВАСЬЄВ Степан……………………………………………….... **35**
АУДИТ ЦИФРОВИХ ПІДПИСІВ ВСТАНОВЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ДРОЖАК Олександр……………………………………………………………………… **38**
ПОЛІНОМІАЛЬНИЙ АЛГОРИТМ ПЕРЕВІРКИ ЧИСЕЛ НА ПРОСТОТУ: ТЕСТ АГРАВАЛА–КАЯЛА–САКСЕНИ

КЛІМ Віталій, ЦАВОЛИК Тарас…………………………………………………….... **44**
АРХІТЕКТУРА СИСТЕМИ БЕЗПЕКИ KUBERNETES

КУЛИНА Сергій………………………………………………………………………… **46**
АНАЛІЗ ЕФЕКТИВНОСТІ ГОМОМОРФНОГО ШИФРУВАННЯ ДЛЯ ЗАХИЩЕНИХ ХМАРНИХ ОБЧИСЛЕНЬ

КУХАРУК Олександр………………………………………………………………….... **48**
РИЗИКИ ТА ВРАЗЛИВОСТІ У СМАРТ–КОНТРАКТАХ

МЕЛЬКО Іванна, ІГНАТЄВ Ігор……………………………………………………..... **51**
РОЗРОБКА ПРОТОТИПУ СИСТЕМИ КЕРУВАННЯ ДОСТУПОМ У БАЗІ ДАНИХ ІЗ ФУНКЦІОНАЛЬНИМ ШИФРУВАННЯМ

МУДРИЙ Іван, БАБАЛА Людмила……………………………………………………..... **53**
ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ БІОМЕТРИЧНОЇ АВТЕНТИФІКАЦІЇ НА ОСНОВІ КРИТЕРІЮ ВІДНОСНОЇ ЕНТРОПІЇ

ОСІДАК Владислав, ІВАСЬЄВ Степан……………………………………………..... **56**
ОНЛАЙН ЗАСОБИ ДИНАМІЧНОГО АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 004.056.5

Владислав ОСІДАК, Степан ІВАСЬЄВ

Західноукраїнський національний університет

ОНЛАЙН ЗАСОБИ ДИНАМІЧНОГО АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ. Актуальність теми "Онлайн засоби динамічного аналізу шкідливого програмного забезпечення" зумовлена стрімким зростанням кількості складних і добре замаскованих кіберзагроз. Сучасні шкідливі програми часто використовують техніки ухилення від виявлення, що робить традиційні методи недостатньо ефективними. Онлайн–платформи динамічного аналізу дозволяють швидко й ефективно досліджувати поведінку підозрілих файлів у контрольованому середовищі. Це робить їх важливим інструментом для оперативного реагування на загрози та покращення кіберзахисту.

Метою дослідження є вивчення можливостей та ефективності онлайн засобів динамічного аналізу для виявлення та дослідження шкідливого програмного забезпечення. Дослідження передбачає аналіз функціональності, переваг і недоліків популярних онлайн–платформ, а також оцінку їх практичного застосування в умовах реальних кіберзагроз. Особлива увага приділяється ролі цих засобів у сучасних системах кіберзахисту та швидкому реагуванні на інциденти.

1. Онлайн–служби для аналізу шкідливого програмного забезпечення.

VirusTotal – безкоштовна служба для аналізу підозрілих файлів та посилань. Подання не потребує.

Intezer – детектор зловредів, що підтримує динамічний та статичний аналіз.

Triage – онлайн–сервіс для аналізу великих обсягів шкідливого програмного забезпечення з функцією статичного аналізу зразків.

FileScan.IO – служба аналізу шкідливих програм, з динамічним аналізом та функцією вилучення індикаторів компрометації (IOC).

Sandbox.pikker – онлайн–версія відомої системи аналізу шкідливих програм Cuckoo Sandbox. Надає докладний звіт з описом поведінки файлу під час виконання у реалістичному, але ізольованому середовищі далеко у хмарі.

Manalyzer – безкоштовний сервіс для статичного аналізу PE–файлів та виявлення маркерів небажаної поведінки. Має офлайн–версію.

Opswat – сканує файли, домени, IP–адреси та хеші за допомогою технології Content Disarm & Reconstruction.

InQuest Labs – сервіс для сканування текстових документів файлів Microsoft та Open Office, електронних таблиць та презентацій. Працює з урахуванням механізмів Deep File Inspection (DFI).

Any Run – ще одна онлайн–пісочниця з гарним інтерфейсом та додатковими опціями.

Yoroi – італійська служба аналізу підозрілих файлів на базі пісочниці. Перетравлює PE (наприклад, .exe–файли), документи (doc і PDF), файли сценаріїв (типу wscript, Visual Basic) та APK, але болісно повільно готує звіти.

Unpacme – онлайн–сервіс для автоматичного розпакування шкідливих програм та вилучення артефактів.

Malwareconfig – веб-додаток для вилучення, декодування та відображення параметрів конфігурації найпоширеніших шкідливих програм.

Malsub – фреймворк Python RESTful для роботи з API онлайн–сервісів для аналізу шкідливого ПЗ.

2. Аналіз можливостей онлайн засобів виявлення шкідливого програмного забезпечення.

Для тестування засобів динамічного аналізу ШПЗ створимо невеликий скрипт мовою python, котрий копіює себе в тимчасову папку та додає в автозавантаження. Скрипт наведений на рисунку 1.

```
import os
import shutil
import sys
import winreg

def add_to_startup(filepath=None, name="MyPythonApp"):
    if filepath is None:
        filepath = sys.executable  # Поточний виконуваний файл (у .exe або .py)

    key = winreg.HKEY_CURRENT_USER
    reg_path = r"Software\Microsoft\Windows\CurrentVersion\Run"

    try:
        reg_key = winreg.OpenKey(key, reg_path, 0, winreg.KEY_SET_VALUE)
    except FileNotFoundError:
        reg_key = winreg.CreateKey(key, reg_path)

    winreg.SetValueEx(reg_key, name, 0, winreg.REG_SZ, filepath)
    winreg.CloseKey(reg_key)

def copy_to_temp():
    temp_dir = os.environ.get("TEMP")
    target_path = os.path.join(temp_dir, "MyPythonApp.exe")

    if not os.path.exists(target_path):
        shutil.copy(sys.argv[0], target_path)
    return target_path

if __name__ == "__main__":
    copied_path = copy_to_temp()
    add_to_startup(filepath=f'python "{copied_path}"')
```

Рисунок 1 – Тестовий скрипт

Спочатку проведемо дослідження за допомогою сервісу https://tria.ge/. Triage – це потужне рішення для автоматизованого аналізу шкідливих програм, розроблене компанією Hatching (ізазначено, що команда створювала Cuckoo Sandbox). Платформа підтримує масштаб до 500 000 аналізів на добу, що робить її оптимальним інструментом для великих організацій і SOC/CERT команд.

На рисунку 2 приведено процес завантаження файлу та вибір віртуальної машини.

Triage дозволяє динамічно досліджувати зразки на таких операційних системах: Windows 7 та 10, Linux, macOS, Android .

Додатково є можливість інтерактивної взаємодії з віртуальною машиною у реальному часі, включаючи ручне введення дій або огляд виконання зразка "вживу" .

Triage автоматично проводить статичний аналіз файлу, видобуває конфігурації шкідливого ПЗ, та генерує динамічні behavioral–оцінки (signature-based, мережеві контакти, доменні репутації), які об'єднуються в загальну оцінку загрози.

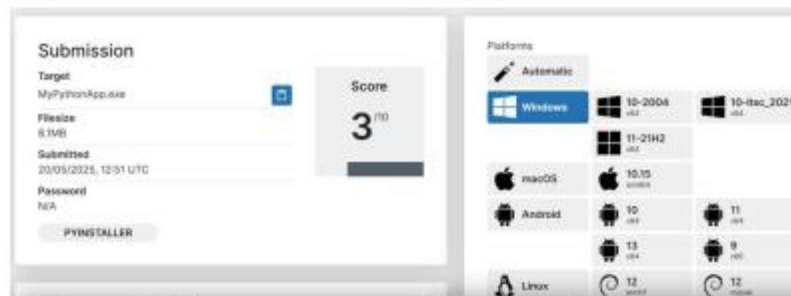

Рисунок 2 – Процес завантаження файлу та вибір віртуальної машини

Інтерфейс з фокусом на найважливішу інформацію: труєтвінг файла, конфігурації, потенційна поведінка, натомість для автоматизації доступна REST API та профілі збереження налаштувань (timeout, інтернет, VM тип) .Результати тестування та роботи програмного засобу можна побачити на рисунку 3. Якщо були якісь повідомлення та виводи на екран, система надасть скрін з екрану в звіт.

Рисунок 3 – Зображення отримані в результаті виконання програмного засобу

Також автоматизована пісочниця надає звіт про системні дії програмного засобу: зміни в реєстрі, зміни в файловій системі, системні виклики, як це показано на рисунку 4.

Плюси Triage: висока масштабованість, підтримка багатьох ОС, інтерактивний live-режим, зручна UI та гнучке API для інтеграції з SOC, MSSP або IR-системами. Обмеження:немає підтримки Windows 11, усі результати аналізів у публічній версії є доступні спільноті, якщо не використовується корпоративний контракт.

Ще одним доступним та зручним засобом є https://www.hybrid–analysis.com/. Hybrid Analysis (що працює на базі CrowdStrike Falcon Sandbox) – це хмарна платформа для статично–динамічного аналізу шкідливого ПЗ, що дозволяє безкоштовно надсилати файли чи URL для дослідження. Основні можливості:

Комбінований аналіз: поєднує статичну перевірку (розбір бінарних структур), динамічне виконання у sandbox–оточенні з моніторингом змін пам'яті, реєстру та мережевих подій.

Рисунок 4 – Звіт про дії програмного засобу

Автоматично згенеровані IOCs: платформа збирає індикатори і включає їх до детальних звітів, доступних через UI чи API Hybrid Analysis.

Інтеграція Threat Intelligence: включає такі сервіси як Criminal IP, Bfore.Ai та CrowdStrike AI-модуль, що покращують аналіз доменів, URL і злочинних IP–адрес. Процес завантаження програмного засобу приведено на рисунку 5.

Рисунок 5 – Завантаження зразка для тесту в Hybrid Analysis

В результаті система сформує звіт по діях зразка та надасть оцінку ризику поведінковим аналізом, як це показано на рисунку 6.

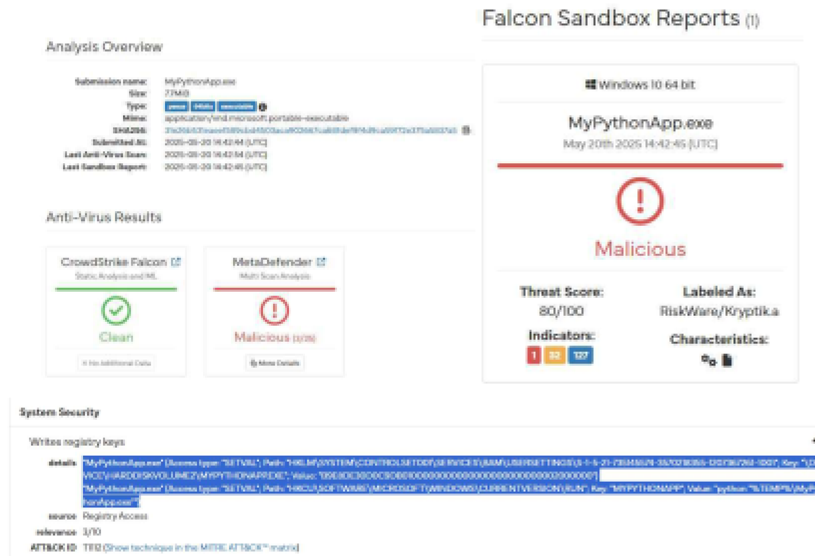

Рисунок 6 – Результат аналізу зразка Hybrid Analysis

Також доступний сервіс для динамічного аналізу ШПЗ https://www.virustotal.com/. VirusTotal – це потужна онлайн–платформа, яка поєднує як статичний, так і динамічний аналіз файлів та URL–адрес з метою виявлення шкідливого програмного забезпечення. Динамічний аналіз у VirusTotal виконується за допомогою внутрішніх sandbox–середовищ, таких як Jujubox для Windows, Droidy для Android та спеціалізованих рішень для macOS. У цих середовищах підозрілі файли запускаються у віртуальних машинах, де система фіксує їхню поведінку: створення нових процесів, зміну реєстру, доступ до мережі, спроби завантаження додаткових компонентів тощо.

Результати динамічного аналізу доступні у вигляді детального звіту, що містить індикатори компрометації (IOC), пов'язані домени та IP–адреси, а також мітки відповідності до тактик MITRE ATT&CK. Такі звіти дозволяють аналітикам швидко оцінити потенційну загрозу та встановити зв'язки з відомими зразками шкідливого ПЗ. VirusTotal також інтегрує зовнішню аналітику – наприклад, від Mandiant або CrowdStrike – що підсилює глибину поведінкової оцінки.

Безкоштовна версія сервісу забезпечує базовий доступ до результатів динамічного аналізу, але всі завантажені зразки стають публічними. У платній версії доступні приватні sandbox–аналізи, API–інтеграція, гнучке управління середовищем виконання та доступ до додаткових аналітичних функцій. Загалом, VirusTotal є універсальним інструментом для початкового та поглибленого аналізу загроз, який широко використовується як фахівцями з кібербезпеки, так і дослідниками з усього світу.

Рисунок 7 – Результати аналізу зразка за допомогою VirusTotal

Висновок. Онлайн засоби динамічного аналізу шкідливого програмного забезпечення є важливим компонентом сучасної системи кіберзахисту. Вони дозволяють швидко й безпечно досліджувати поведінку підозрілих програм у ізольованому середовищі, що значно підвищує ефективність виявлення нових та складних загроз.

Попри деякі обмеження, такі як залежність від інтернет–з'єднання чи обмежена функціональність безкоштовних сервісів, ці інструменти демонструють високу практичну цінність. У майбутньому їх розвиток і інтеграція з іншими аналітичними системами сприятимуть ще більш ефективному реагуванню на кіберінциденти.

Перелік використаних джерел.

1. Aslan Ö., Samet R., "Investigation of Possibilities to Detect Malware Using Existing Tools," 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, 2017, pp. 1277–1284, doi: 10.1109/AICCSA.2017.24.

2. Saurabh, "Advance Malware Analysis Using Static and Dynamic Methodology," 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), Bhopal, India, 2018, pp. 1–5, doi: 10.1109/ICACAT.2018.8933769.

3. D. Kim, D. Mirsky, A. Majlesi–Kupaei and R. Barua, "A Hybrid Static Tool to Increase the Usability and Scalability of Dynamic Detection of Malware," 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), Nantucket, MA, USA, 2018, pp. 115–123, doi: 10.1109/MALWARE.2018.8659373.

4. Sinha A. K., Sai S., "Integrated Malware Analysis Sandbox for Static and Dynamic Analysis," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1–5, doi: 10.1109/ICCCNT56998.2023.10306805.

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2025)

науково-практична конференція
молодих вчених, аспірантів та студентів

28–29 серпня 2025
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2025), Тернопіль, 2025. - 154 с.

Редакційна колегія:

Василь ЯЦКІВ – доктор технічних наук, професор, завідувач кафедри кібербезпеки, Західноукраїнський національний університет.

Михайло КАСЯНЧУК – доктор технічних наук, професор, професор кафедри кібербезпеки, Західноукраїнський національний університет.

Ігор ЯКИМЕНКО – кандидат технічних наук, доцент, декан факультету комп'ютерних інформаційних технологій, Західноукраїнський національний університет.

Лідія ТИМОШЕНКО – кандидат економічних наук, доцент, завідувач кафедри кібербезпеки та програмного забезпечення, Національний університет «Одеська політехніка».

Наталя СТЕФУРАК – кандидат фізико-математичних наук, завідувач відділенням комп'ютерних технологій, Галицький фаховий коледж ім. В'ячеслава Чорновола.

Наталія ЯЦКІВ – кандидат технічних наук, доцент, доцент кафедри спеціалізованих комп'ютерних систем, Західноукраїнський національний університет.

Степан ІВАСЬЄВ – кандидат технічних наук, доцент, доцент кафедри кібербезпеки, Західноукраїнський національний університет.

Тарас ЦАВОЛИК – кандидат технічних наук, доцент, доцент кафедри кібербезпеки, Західноукраїнський національний університет.

Людмила БАБАЛА – кандидат економічних наук, доцент, доцент кафедри кібербезпеки, Західноукраїнський національний університет.

Сергій КУЛИНА – PhD, доцент кафедри кібербезпеки, Західноукраїнський національний університет.

Ігор ІГНАТЄВ – викладач кафедри кібербезпеки, Західноукраїнський національний університет.

Аліна ДАВЛЕТОВА – викладач кафедри кібербезпеки, Західноукраїнський національний університет.

Головний редактор: Михайло КАСЯНЧУК
Технічний редактор: Аліна ДАВЛЕТОВА

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
e-mail:conferencekb@gmail.com

ЗМІСТ

СИСТЕМИ ТА ТЕХНОЛОГІЇ КІБЕРБЕЗПЕКИ

Ярова Інна, Власова Аліса, Кушніренко Наталія
АНАЛІЗ НОРМАТИВНОЇ БАЗИ ДЛЯ СТВОРЕННЯ МОДЕЛІ ПОРУШНИКА ІНФОРМАЦІЙНОЇ БЕЗПЕКИ **7**

Юр'єв Д.А., Тимошенко Л.М.
КІБЕРСИТУАЦІЙНА ОБІЗНАНІСТЬ СПІВРОБІТНИКІВ ОБ'ЄКТУ КРИТИЧНОЇ ІНФРАСТРУКТУРИ **9**

Чабаненко К.С., Бобок І.І., Кушніренко Н.І.
МОДЕЛЬ CYBERCRIME-AS-A-SERVICE В СУЧАСНОМУ ЛАНДШАФТІ КІБЕРЗАГРОЗ **12**

Шамарін В.В., Вінковська І.С.
БЕЗПЕЧНИЙ ОБМІН ДАНИМИ В ДЕЦЕНТРАЛІЗОВАНИХ P2P-СИСТЕМАХ **15**

Власова А.С., Кушніренко Н.І., Назарова І.В.
АЛГОРИТМ ТЕКСТОВОГО АНАЛІЗУ ДЛЯ ПРОФІЛЮВАННЯ КОРИСТУВАЧІВ В OSINT ДОСЛІДЖЕННЯХ **17**

Пянковська Вікторія, Ярова Інна
СУЧАСНІ МЕТОДИ ТЕЛЕФОННОГО ТА ОНЛАЙН-ШАХРАЙСТВА В УКРАЇНІ: МЕТОДІ ПРОТИДІЇ ТА РОЗКРИТТЯ ЗЛОЧИНІВ **20**

Завадський Д.О., Кушніренко Н.І.
РОЗРОБКА НАВЧАЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПРОТИДІЇ АТАКАМ СОЦІАЛЬНОЇ ІНЖЕНЕРІЇ **23**

Бевз Валентин
АНАЛІЗ АКТУАЛЬНИХ ВРАЗЛИВОСТЕЙ MS OFFICE **25**

Лаковський Б.А., Сиропятов О.А., Тимошенко Л.М.
ПОТОЧНИЙ СТАН ТА ПРОБЛЕМАТИКА ВПРОВАДЖЕНЯ ЗАХИСТУ ІНФОРМАЦІЇ У ДЕРЖАВНИХ ПРОМИСЛОВИХ СИСТЕМАХ **28**

Сегеда Євген, Давлетова Аліна
КОМБІНОВАНА СИСТЕМА МОНІТОРИНГУ ТА ВИЯВЛЕННЯ MALWARE-ЗАГРОЗ **31**

Назаров В.О.
АВТОМАТИЗОВАНИЙ МЕТОД РИЗИК-ОРІЄНТОВАНОГО ВИЯВЛЕННЯ ПРОБЛЕМНИХ ПРОФІЛІВ У СОЦМЕРЕЖАХ **35**

Драгін Д., Садченко А.
РОЗРОБКА ЛОКАЛЬНОЇ МОДЕЛІ МАШИННОГО НАВЧАННЯ ЩОДО ЗАХИСТУ КОНФІДЕНЦІЙНОЇ ІНФОРМАЦІЇ У ВІДКРИТОМУ ПРОГРАМНОМУ КОДІ **38**

УДК 004.056.5

Владислав ОСІДАК

Західноукраїнський національний університет

ПОВЕДІНКОВИЙ АНАЛІЗ У ЗАДАЧІ ВИЯВЛЕННЯ ШКІДЛИВИХ ПРОГРАМ

Вступ. Актуальність теми "Поведінковий аналіз у задачі виявлення шкідливих програм" обумовлена зростаючими загрозами кібербезпеки та складністю виявлення нових типів шкідливого ПЗ. Традиційні методи, які базуються на сигнатурах, часто не здатні ефективно розпізнати нові або модифіковані віруси.

Поведінковий аналіз дозволяє відстежувати дії програм у реальному часі, що дає змогу виявляти шкідливі програми навіть до їх поширення. Це робить методи поведінкового аналізу важливими для підвищення ефективності захисту від кіберзагроз.

Метою дослідження є аналіз ефективності застосування методів поведінкового аналізу для виявлення шкідливих програм. Дослідження також має на меті виявити основні переваги та обмеження цього підходу порівняно з традиційними методами. Окрім того, буде розглянуто можливості інтеграції поведінкового аналізу в сучасні системи кіберзахисту для покращення виявлення та нейтралізації нових загроз.

1. Підходи до аналізу шкідливого програмного забезпечення

Шкідливе програмне забезпечення вже довгий час є однією з основних загроз в галузі інформаційної безпеки. Підходи до аналізу та захисту від таких атак бувають різні. Загалом поділяють два підходи: статичний та динамічний аналіз.

Завдання статичного аналізу - пошук шаблонів шкідливого вмісту у файлі чи пам'яті процесу. Це можуть бути рядки, фрагменти закодованих або стислих даних, послідовності компільованого коду. Може здійснюватися пошук як окремих шаблонів, а й їх комбінацій з додатковими умовами (наприклад, з прив'язкою до місця знаходження сигнатури, перевіркою відносної відстані в розміщені один від одного).

Динамічний аналіз – це аналіз поведінки програми. Варто зазначити, що програма може бути запущена в так званому емульованому режимі. Передбачається безпечне інтерпретування дій без завдання пошкоджень операційній системі. Інший спосіб - запуск програми у віртуальному середовищі (пісочниці). У такому разі буде чесне виконання дій на системі з подальшою фіксацією дзвінків. Ступінь подробиці логування - це свого роду баланс між глибиною спостереження та продуктивністю аналізуючої системи. На виході виходить журнал дій програми операційній системі (траса поведінки), який піддається подальшому аналізу.

Динамічний чи поведінковий аналіз дає ключову перевагу - незалежно від спроб заплутування програмного коду та прагнень приховати наміри зловмисника від вірусного аналітика шкідливий вплив буде зафіксовано. Зведення завдання

виявлення ВПО до аналізу дій дозволяє висунути гіпотезу про стійкість просунутого алгоритму виявлення шкідливих даних. А відтворюваність поведінки, завдяки тому самому початковому стану середовища для аналізу (зліпка стану віртуального сервера), спрощує вирішення завдання класифікації легітимного і шкідливого поведінки.

Часто підходи у поведінковому аналізі ґрунтуються на наборах правил. Експертний аналіз переноситься в сигнатури, на основі яких інструмент детекту шкідливого ПЗ та файлів робить висновки. Однак у такому разі може виникнути проблема: можуть враховуватися лише ті атаки, які суворо відповідають написаним правилам, а атаки, які не виконують ці умови, але все ще шкідливі, можна пропустити. Та ж проблема виникає у разі змін одного й того ж шкідливого ПЗ. Вирішити це можна за допомогою більш м'яких критеріїв спрацьовування, тобто можна написати більш загальне правило, або за допомогою великої кількості правил під кожен шкідливість. У першому сценарії ми ризикуємо отримати багато помилкових спрацьовувань, а другий вимагає серйозних витрат за часом, що може призвести до запізнення необхідних оновлень.

З'являється потреба у поширенні вже наявних знань інші схожі випадки. Тобто ті, які раніше ми не зустрічали і не обробляли правилами, але на основі схожості деяких ознак можемо зробити висновок, що активність може бути шкідливою. Тут і допомагають алгоритми машинного навчання.

ML-моделі під час коректного навчання мають узагальнюючу здатність. Це означає, що навчена модель не просто вивчила всі приклади, на яких навчалася, а здатна приймати рішення для нових прикладів на основі закономірностей із навчальної вибірки.

Однак для того, щоб узагальнююча здатність працювала, необхідно враховувати два основні фактори на етапі навчання:

Набір ознак повинен бути якомога повнішим (щоб модель могла бачити якнайбільше закономірностей, відповідно, краще поширювала свої знання на нові приклади), але не надлишковим (щоб не зберігати і не обробляти ознаки, які не несуть у собі корисну інформацію для моделі).

Набір даних має бути репрезентативним, збалансованим та регулярно оновлюваним.

2. Процес переносу експертного знання в моделі машинного навчання

У контексті аналізу шкідливого програмного забезпечення вихідні дані - це самі файли, а проміжні дані - це створені ними допоміжні процеси. Процеси, у свою чергу, здійснюють системні виклики. Послідовності таких викликів є дані, які нам необхідно перетворити на набір ознак.

Складання датасету розпочалося на експертній стороні. Було обрано ознаки, які, на думку експертів, мають бути значущими з погляду виявлення ШПЗ. Усі ознаки можна було звести до виду n-грам за системними викликами.

Далі за допомогою моделі проведена оцінка, тих ознак які роблять найбільший внесок у виявлення, відкинули зайве і отримали підсумкову версію датасета.

Вихідні дані:

{"count":1,"PID":"764","Method":"NtQuerySystemInformation","unixtime":"1639557419.628073","TID":"788","plugin":"syscall","PPID":"416","Others":"REST: ,Module=\"nt\",vCPU=1,CR3=0x174DB000,Syscall=51,NArgs=4,SystemInformationClass=0x53,SystemInformation=0x23BAD0,SystemInformationLength=0x10,ReturnLength=0x0","ProcessName":"windows\\system32\\svchost.exe"}

{"Key":"\\registry\\machine","GraphKey":"\\REGISTRY\\MACHINE","count":1,"plugin":"regmon","Method":"NtQueryKey","unixtime":"1639557419.752278","TID":"3420","ProcessName":"users\\john\\desktop\\e95b20e76110cb9e3ecf0410441e40fd.exe","PPID":"1324","PID":"616"}

{"count":1,"PID":"616","Method":"NtQueryKey","unixtime":"1639557419.752278","TID":"3420","plugin":"syscall","PPID":"1324","Others":"REST: ,Module=\"nt\",vCPU=0,CR3=0x4B7BF000,Syscall=19,NArgs=5,KeyHandle=0x1F8, KeyInformationClass=0x7,KeyInformation=0x20CD88,Length=0x4,ResultLength=0x20CD98","ProcessName":"users\\john\\desktop\\e95b20e76110cb9e3ecf0410441e40fd.exe"}

3. Покращення якості моделі з кожним оновленням

Вважаємо вибірку найбільш коректною, тому що приклади цієї вибірки перевіряються і розмічуються експертами вручну, і з кожним оновленням перевіряється в першу чергу те, що гарантується 100% точності на цій вибірці. Тестування in the wild підтверджує, що точність покращується.

Досягається це за рахунок очищення навчальної вибірки від еталонних даних, що суперечать. Під даними, що суперечать, ми розуміємо приклади, накопичені з потоку, які досить близькі по векторній відстані до трас з еталонної вибірки, але при цьому мають протилежну мітку.

Експерименти показали, що такі приклади є викидами навіть з погляду даних із потоку, оскільки після видалення їх із навчальної вибірки з метою підвищення точності на еталонній вибірці, зростала і точність на потоці.

4. Взаємодоповнення ML-підходу та поведінкових детектів у вигляді кореляцій

ML-модель дуже добре проявила себе у поєднанні з поведінковими детектами у вигляді кореляцій. Важливо зауважити, що саме в поєднанні, так як узагальнююча здатність моделі хороша у випадках, коли необхідно розширити рішення виявленням схожих та близьких інцидентів, але не у випадках, коли потрібен детект у рамках чіткого розуміння правил та критеріїв того, що є шкідливим ПЗ.

Прикладами, де ML-підхід зміг дійсно розширити рішення, стали:

– Аномальні ланцюжки підпроцесів. Саме собою велика кількість гіллястих ланцюжків - явище легітимне. Але аномальність у кількості вузлів, ступеня вкладеності, повторюваності чи повторюваності якихось конкретних імен процесів модель зауважує, а людина заздалегідь таке не нафантазує знайти шкідливим.

– Нестандартні параметри дзвінків за промовчанням. Найчастіше

аналітика цікавлять значні параметри функцій, у яких шукають ШПЗ. Інші параметри, грубо кажучи, значення за замовчуванням, вони не особливо цікавлять. Але в якийсь момент так виходить, що замість припустимо п'яти значень за умовчанням зустрічається шосте. Аналітик міг припустити, що таке можливо, а модель помітила.

– Нетипові послідовності викликів функцій. Той випадок, коли кожна функція окремо робить нічого шкідливого. Та й разом - теж. Але так сталося, що їхня послідовність не зустрічається в легітимному ПЗ. Аналітику буде потрібний гігантський досвід, щоб самостійно помітити таку закономірність. А модель помічає (і не одну), вирішуючи нестандартно завдання класифікації за ознакою, яка взагалі не закладалася як показник шкідливості.

Використання конкретного компонента одним викликом для шкідливої дії. Система використовує сотні об'єктів у різній варіативності, різною мірою. Вловити використання одного на тлі мільйона інших навряд чи вдасться - гранулярність аномалії все ж таки занизька. Проактивний детект за моделлю загроз. Вирішили, що певний вплив на певний об'єкт у системі хоча б один раз, неприпустимо. Модель може з першого разу не зрозуміти, що це значуще явище і буде шанс помилки чи невпевненого рішення на етапі класифікації чогось схожого. Обфускація послідовності процесів. Наприклад, може бути відомо, що потрібно зробити 3-4 дії у визначеному порядку. Не має значення, що буде між ними. Якщо накидати випадкові дії між 3-4 ключовими - це модель, рішення буде прийнято неправильно. При цьому розмірність числа ознак не дозволяє враховувати такі заплутування зберігання всіх комбінацій послідовностей викликів, а не тільки загальної кількості.

Висновок. Поведінковий аналіз є ефективним інструментом для виявлення шкідливих програм, оскільки дозволяє виявляти нові та модифіковані загрози, яких не можна розпізнати за допомогою традиційних методів. Він забезпечує гнучкість та адаптивність у боротьбі з кіберзагрозами, враховуючи динамічний характер сучасних вірусів. Однак для досягнення максимальної ефективності необхідно інтегрувати поведінковий аналіз із іншими методами захисту. У результаті, використання цього підходу може значно підвищити рівень безпеки в цифрових середовищах.

Перелік використаних джерел.

1. A. A. Selçuk, F. Orhan and B. Batur, "Undecidable problems in malware analysis," 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), Cambridge, UK, 2017, pp. 494-497, doi: 10.23919/ICITST.2017.8356458.

2. A. Afreen, M. Aslam and S. Ahmed, "Analysis of Fileless Malware and its Evasive Behavior," 2020 International Conference on Cyber Warfare and Security (ICCWS), Islamabad, Pakistan, 2020, pp. 1-8, doi: 10.1109/ICCWS48432.2020.9292376.

3. O. Or-Meir, A. Cohen, Y. Elovici, L. Rokach and N. Nissim, "Pay Attention: Improving Classification of PE Malware Using Attention Mechanisms Based on System Call Analysis," 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 2021, pp. 1-8, doi: 10.1109/IJCNN52387.2021.9533481.